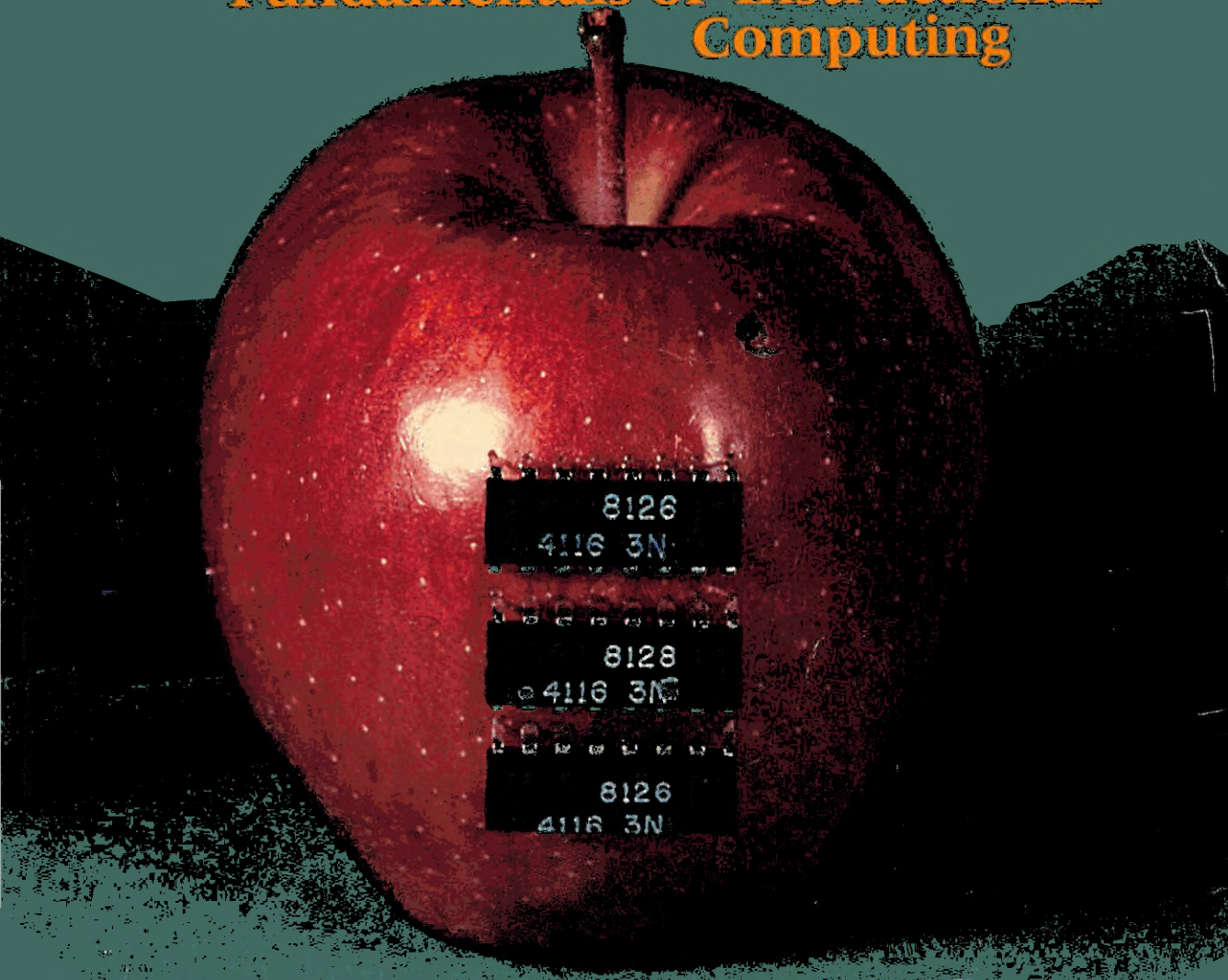
AN APPLE FOR THE TEACHER

Fundamentals of Instructional Computing



George Culp Herbert Nickles

An Apple for the Teacher

Brooks/Cole Series in Computer Science

Program Design with Pseudocode
T. E. Bailey and K. A. Lundgaard

BASIC: An Introduction to Computer Programming with the Apple
Robert J. Bent and George C. Sethares

BASIC: An Introduction to Computer Programming, Second Edition
Robert J. Bent and George C. Sethares

Business BASIC
Robert J. Bent and George C. Sethares

FORTRAN with Problem Solving: A Structured Approach
Robert J. Bent and George C. Sethares

Beginning BASIC
Keith Carver

Beginning Structured COBOL
Keith Carver

Structured COBOL for Microcomputers
Keith Carver

Learning BASIC Programming: A Systematic Approach
Howard Dachslager, Masato Hayashi, and Richard Zucker

Problem Solving and Structured Programming with ForTran 77
Martin O. Holoien and Ali Behforooz

Basic Business BASIC: Using Microcomputers
Peter Mears and Louis Raho

Brooks/Cole Series in Computer Education

An Apple for the Teacher: Fundamentals of Instructional Computing
George H. Culp and Herbert Nickles

RUN: Computer Education
Dennis O. Harper and James H. Stewart

An Apple for the Teacher

Fundamentals of Instructional Computing

George H. Culp
Assistant Director for Instructional Computing
Computation Center
University of Texas at Austin

Herbert Nickles
Coordinator of Instructional Computing
Computer Center
California State College, San Bernardino

Brooks/Cole Publishing Company
Monterey, California

Brooks/Cole Publishing Company
A Division of Wadsworth, Inc.

Printed in the United States of America

10 9 8 7 6 5 4 3

Library of Congress Cataloging in Publication Data

Culp, George H.
An Apple for the teacher.

Bibliography: p.
Includes index
1. Computer-assisted instruction—Teacher training.
2. Computer managed instruction—Teacher
training. 3. Microcomputers—Teacher training.
I. Nickles, Herbert. II. Title.
LB1028.5.C78 1983 371.3'9445 82–24506

ISBN 0-534-01378-3

This product is neither endorsed nor coproduced by Apple Computer, Inc.

Subject Editor: James F. Leisy, Jr.
Production Service: Greg Hubit Bookworks
Manuscript Editor: Trevor Grayling
Interior Design: Marilyn Langfeld
Cover Design: Vicki Van Deventer
Cover Photo: Stan Rice
Typesetting: Graphic Typesetting Service, Los Angeles

Preface

"The PTA at our school has given us two microcomputers. Since I am a math/science teacher, the principal said I should use them in my classes. I had one course in Fortran programming about five years ago; I know very little about computers and absolutely nothing about how to use them with instruction. I am aware that they are being used as effective teaching tools. But how? What do I need to know to get started?"

An Inservice Teacher

"As part of an education course I'm taking, I recently sat in on a school board meeting of our district. The agenda included a presentation by a man demonstrating the instructional uses of microcomputers. The man's assistant, who operated the microcomputer, loaded and ran programs and, in general, demonstrated the system, was his six-year-old daughter! It is obvious to me that I'll be facing many young students who are using microcomputers at home and in the school. Even more obviously, I'll need to know something about the different uses of micros: What they can and cannot do, how they are programmed, how these programs are made and tested, and so on. But I don't want to become a "computer scientist." I want to know the fundamentals that will let me make practical use of a microcomputer in an instructional setting."

A Preservice Teacher

This book is designed for teachers who find themselves in situations similar to those cited in the two examples above. It is based upon a university course in which hundreds of teachers in grade levels elementary through college have been introduced to the fundamentals of the instructional use of computers and have successfully designed and developed programs for use in their own areas of interest.

This book is not designed to teach general computer literacy: There is little mention of the history, architecture, or use of computers in society. Nor is it a text to train computer programmers: Several language statements common to programming texts are omitted because their application is not typical of the *instructional* use of computers.

This is a practical book for the teacher who needs to know the *fundamentals* of the BASIC programming language for the Apple microcomputer and how to apply them to the design and development of instructional computing programs. It has been our experience that, given these fundamentals, teachers have the proficiency to expand upon this base and develop efficient programs designed to meet their specific needs.

The text consists of nine chapters and four appendices and is divided into two parts. In Part One, the first four chapters discuss the BASIC programming language statements and commands common to five areas of instructional computing use: problem solving, drill and practice, tutorial dialog, simulation and gaming, and testing. Chapter 5 summarizes and reviews these statements and their applications. Chapter 6 gives relatively short example and model programs in each of these five areas. Chapter 7 discusses and demonstrates the simple use of graphics as an instructional technique.

In Part Two, Chapters 8 and 9 discuss the specific steps needed to first design and then develop instructional computing programs. The appendices include instructions for "booting up" the microcomputer; instructions for loading, editing, and saving programs; commands and statements unique to the Apple microcomputer; answers to questions and problems given in the chapters; and an annotated bibliography of journals and other publications dealing with instructional computing.

As a matter of personal preference, some readers may wish to study Chapters 8 and 9 on design and development prior to the chapters on BASIC. We believe, however, that practical design and development can come only after the working guidelines for the language are established. Thus, BASIC fundamentals are presented before the discussion of design fundamentals.

Twenty-six programs ranging from simple introductory examples to more complex instructional computing application models, plus a "keyword" subroutine and a program "menu" routine are presented in the text. Professors adopting the text may write to the publisher for a free copy of the software diskette, which contains these programs along with solution programs to selected problems in the text. On request, the publisher will make copies of the diskette available to students for $11.95 each.

The authors are indebted to many people for the development of this book. Only with the critical review of the manuscript by Sister Mary K. Keller of Clark College, Professor Edward B. Wright of Western Oregon State College, Professors Dennis Harper and Jeffrey Marcus of the University of California at Santa Bar-

bara, the encouragement of Mr. James F. Leisy, Jr., of Brooks/Cole Publishing Company, and the excellent copy editing of Mr. Trevor Grayling, could this book have been published. Production was ably directed by Mr. Greg Hubit of Greg Hubit Bookworks. Additional assistance was provided by Mr. Morgan Watkins of the University of Texas at Austin, Mr. Carey Van Loon of California State College, San Bernardino, and Mr. Larry Hall. Most of all, we wish to gratefully acknowledge the contribution of the 708 students who have provided direct input and response during the development stages of this book.

GHC
HN

Contents

Introduction

This book contains nine chapters that describe an approach to using a common programming language, BASIC, for the design and development of instructional computing programs for Apple microcomputers. These chapters discuss certain fundamentals of the language and the design and developmental processes that provide a foundation for the production of instructional computing programs.

There are more than one hundred books available that teach BASIC (the *B*eginners' *A*ll-purpose *S*ymbolic *I*nstruction *C*ode, developed by John G. Kemeny and Thomas E. Kurtz at Dartmouth College). Although most of these books are very thorough in describing BASIC, they usually emphasize problem-solving applications. Our emphasis, on the other hand, is on instruction in the use of BASIC in the design and development of materials for *instructional computing*, which we now proceed to define.

Simply put, any use of computing techniques within the classroom may be broadly defined as instructional computing (sometimes known as *computer-assisted instruction*). Specifically, it includes:

1. *Problem solving*, in which computer programs are written to solve discipline-oriented problems.
2. *Drill and practice* on fundamental concepts using computer programs in a given discipline.
3. *Tutorial dialog*, in which computer programs provide "tutorlike" assistance in pointing out certain types of mistakes, providing review if needed, skipping areas in which proficiency is shown, and so on.
4. *Simulation*, in which computer programs allow manipulation and interpretation of certain elements related to given physical or social phenomena without the constraints of time, space, equipment, and environmental or logistical limits.
5. *Testing*, in which computer programs ask the questions, check the answers, and record the performance.

For our purposes, the term *instructional computing* is used to include all of these applications.

The Use of BASIC An introduction to some of the fundamentals of BASIC is provided in this book. This introduction is not intended to produce highly accomplished and skilled programmers. Rather, it gives only the fundamentals needed to write fairly simple programs for instructional computing applications. Model programs are described that illustrate this use.

Although many different programming languages may be used in instructional computing, there are several reasons for using BASIC:

1. It is easy to learn and easy to use.
2. It is a common interactive language (see Section 1.3), available on large computer systems costing millions, medium-sized systems costing hundreds of thousands, minisystems costing tens of thousands, and small systems (commonly called *micros* or *personal* computers) costing a few hundred to a few thousand dollars.
3. It may be used in all applications of computer-based instruction.
4. It is the introductory computer language used in most secondary and many elementary schools.
5. It is the most common language of microcomputers—an area of computer technology that is making the major impact on education in this decade.

Design Following the introduction to BASIC, a method for designing instructional materials called the *systems approach* is outlined. This approach, in essence, is a logical, step-by-step process for identifying the tasks and activities needed in the production of validated instructional materials.

Development The development of instructional computing programs by the reader is the ultimate goal of this book. Initially, the development phase overlaps the design phase in which paper, pencil, and brain power are the principal ingredients. This involves outlining the rationale, objectives, and instructional sequence of one or more instructional computing programs. After this is outlined on paper, it is translated into the BASIC programming code. Following this, it is necessary to spend considerable time at a computer entering, testing, and refining what has been designed and developed on paper.

As a final introductory note, it should be emphasized that this book assumes no previous experience whatsoever with computers. On the other hand, it is not designed to provide detailed information on computers in general or how they operate. Rather, it introduces the ways and means by which the Apple® computer* may be used within the instructional process.

Now, let us begin by getting down to the BASICs . . .

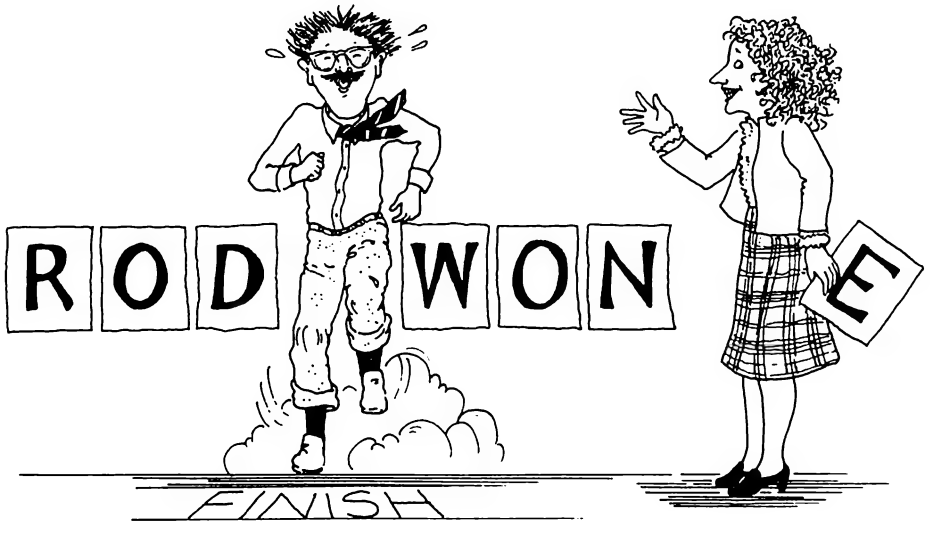
*Apple is a registered trademark of Apple Computer, Inc.

Part I

An Introduction to the BASIC Programming Language

"Nothing in life is to be feared. It is only to be understood."
—Marie Curie

"In certain trying circumstances, urgent circumstances, desperate circumstances, profanity furnishes a relief denied even to prayer."
—Mark Twain



Think About This (for Fun)

Rearrange the letters of NEW DOOR to form one word. [Note: Answers to *Think About This for Fun* questions may be found in Appendix C.]

Think About This (Seriously)

Does a computer possess intelligence?

Chapter 1

A BASIC Program of My Very Own

1.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. List five general applications of computer-based instruction (Introduction).
2. Define two ways in which computers may be accessed (Section 1.3).
3. List the steps necessary to "boot up" (power up) a computer system (Appendix A).
4. State how a BASIC program may be entered on that system after the booting up (Section 1.5.5 and Appendix A).
5. Define what (not who) composes a BASIC program (Section 1.4.1).

6. Distinguish between BASIC statements and commands (Sections 1.4.1–1.4.2).
7. Define the action of the following BASIC commands: NEW, RUN, LIST, and SAVE (Section 1.4.2).
8. Define and give at least one example of both a *Numeric* variable and a *String* variable (Section 1.4.3).
9. Describe the use of commas and semicolons in BASIC for purposes other than punctuation (Section 1.4.4).
10. Define the purpose and give at least one example of the following BASIC statements: PRINT, INPUT, LET, and END (Sections 1.5.1–1.5.4).
11. Describe three simple techniques for editing BASIC programs (Section 1.6 and Appendix B).

1.2 COMPUTER USE: A BRIEF HISTORY AND RATIONALE

Electronic computers have been in use since the late 1940s. In the period from 1948 to 1965, they were used primarily for what their name implies: computing or "number crunching" as it is sometimes called. Starting about the mid-sixties, however, educators began experimenting with applications of computers in the instructional process that involved more than just computing.

In the decade following, this use expanded, and, just as computers have become ingrained in our society, instructional computing is becoming commonplace in our schools. (These points may be emphasized by the fact that since 1975 over 1,500,000 microcomputers have been purchased, many for home or school use.)

Now, it is very important to recognize that computers are not replacing teachers! The fundamental principle underlying the use of computers—regardless of the profession using them—is that they are incredibly fast and accurate tools that allow people to do certain activities in a manner not previously possible. Thus, the use of computers in instruction is basically that of *supplemental* applications. Computers allow teachers and students to do certain educational processes faster, with greater accuracy, and in a manner not possible before they came on the scene.

Computer programs can be very helpful in providing patient, routine drill on fundamental concepts, in generating and grading tests in a given discipline, and in many other applications. In any of these cases, the most effective programs are those designed by teachers—the professionals in the field who are aware of what is to be taught and how to teach it. As yet, there is no computer program that can lead an intelligent and sensitive discussion on any given abstract concept. There are no teachers out of a job because they have been replaced by a computer! That is something worth remembering.

1.3 ACCESS TO COMPUTERS

A computer is an extremely fast and accurate processor of data. In the simplest sense, most common computer systems may be viewed as four units connected electronically:

1. An *input* unit (such as a computer terminal keyboard) through which data is entered.
2. A *processor* unit, which stores the data input and processes it electronically.
3. An *output* unit (such as a computer terminal screen or printer) which shows the results of processing the data input.
4. A *data storage/retrieval* unit (such as a disk drive) which stores data on, and retrieves data from, some magnetic medium (such as a floppy disk).

Figure 1.1 shows these units in block form.

Until the late 1960s, the primary means of access involved punching program statements, data, and commands onto computer cards. This "batch" of cards was read (input) by a card reader and eventually a printout (output) of the program "run" was retrieved. This type of access is commonly referred to as *batch access* or *batch processing*.

Since the early 1970s, there has been a very strong trend toward accessing computers via computer terminals. In the simplest sense, a terminal consists of a keyboard, similar to that of a typewriter, for input of statements, data, commands, and so forth, with output displayed either on a cathode ray tube (CRT) screen or paper (hardcopy) at the terminal. This type of access is known as *interactive* (a user is interacting directly with the computer or a program) or *timesharing* (there may be literally scores of terminals in remote locations "shar-

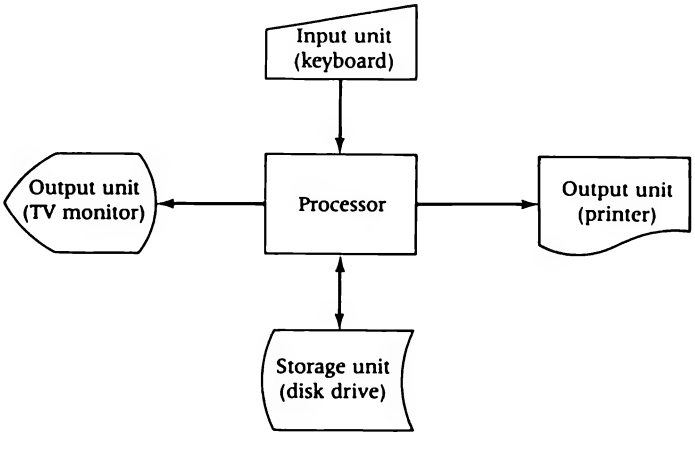


Figure 1.1
Components of a computer system.

Figure 1.2
APPLE II system: keyboard, monitor, thermal printer, and disk drives. (Courtesy of Apple Computer, Inc.)

ing the time" of one computer). In these cases, the terminal is connected to the computer via standard telephone lines.

Microcomputers are an exception to this. Here, the computer, terminal, display, and other components are usually provided as a unit small enough to fit on a desk top (Figure 1.2). There are no telephone connections or sharing of computer time. This makes the unit more portable, less prone to equipment failure, less expensive, and, consequently, well suited to the classroom.

For our use here, only microcomputers are discussed. The examples and assignments in the text assume that the reader has access to an APPLE II microcomputer with AppleSoft BASIC, one floppy disk drive, a video monitor or television, and at least 48K of random access memory (RAM).

It is very important that the reader, particularly the reader new to microcomputers, become familiar with the processes needed to access (use) the system. This first involves gaining confidence in booting up the system. Refer to Appendix A for a step-by-step procedure to accomplish this.

1.4 A BIT ABOUT BASIC BEFORE BEGINNING

There are a few general points about BASIC that should be made early. Consider these as some of the "rules of the game" to follow for BASIC.

1.4.1 Statements

A BASIC program is composed of BASIC statements. These are words (often verbs), such as PRINT, INPUT, and so on, that make some degree of sense to both a user and the computer. (Of course, the computer has been programmed by people to "understand" these words.)

BASIC statements are always numbered, generally by tens (10, 20, 30, etc.). They could be numbered 1, 2, 3, and so on, but no additional statements could be inserted into the program, say, between statements numbered 1 and 2. Statements can be inserted between lines numbered 10 and 20 (11, 12, etc.), and so it is possible to add as many as 9 lines (statements) between 10 and 20. Thus, the numbering convention is usually in increments of ten.

1.4.2 Commands

BASIC commands issue specific information to the computer system about the program. For example, the command NEW instructs the system to prepare for a new BASIC program to be entered at the terminal by "erasing" any program statements that are currently in the system's memory. The command LIST will produce a listing of the BASIC statements comprising the program in memory.

The command RUN executes (RUNs) the BASIC statements in their increasing numerical sequence unless one of those statements transfers the execution to another part of the program. (This is called *branching* and will be discussed later.) The command SAVE ⟨filename⟩ instructs the system to save the program in memory under the name ⟨filename⟩. The program is stored on a floppy diskette placed in the disk drive. (The ⟨filename⟩ may be just about any name the user wishes, but short, descriptive names should be considered.)

1.4.3 Variables

Nearly all BASIC programs described in this text will include values that may *vary* as the program is executed (RUN). These values, which are called *variables*, could be students' names, test scores, responses for correct or incorrect answers, and so forth.

In BASIC, a variable may be represented (named) by any letter of the alphabet (A–Z) or any letter and any number up to 9 (A1,M8,W3,Z9, etc.). The APPLE microcomputer will allow variables to have even longer names that are more descriptive of what they represent: FIRSTTEST, NUMBEROK, AVERAGE, and so on. However, only the first two letters are used internally by the APPLE system. Consequently, AVERAGE and AVENUE represent the same variable; so care must be used in naming the variables in a program.

For our purposes, there are two types of variables:

1. *Numeric.* The value of the variable is always numeric: 1.0, 2, 110.5, −3.1365, and so on.
2. *String (or alphanumeric).* The value of the variable may be alphabetic characters or numbers or a mixture of both. This value is *always* enclosed in quotation marks: "ABCDEF", "CS395T", "JOHN JONES", "NOW IS THE TIME", and so on.

A dollar sign ($) is added to the name of the string variable to distinguish it from a numeric variable. N$, A1$, Z9$, and FIRSTNAME$ all represent string variable names, while N, A1, Z9, and FIRSTTEST all represent numeric variable names.

Examples:
```
A = 123
```
(The *numeric* variable named A has a value of 123.)

```
A$ = "ABC"
```
(The *string* variable named A$ has a value of ABC.)

1.4.4 Commas (,) and Semicolons (;)

Commas and semicolons have specific uses in BASIC. They can be used in the normal fashion as punctuation marks, *or* they can be used to instruct the system to display information in special ways. For example, every so often in a BASIC program there may be a need to have information printed in columns. Suppose a list of student names, test score averages, and final numeric grades were to be displayed (printed). Assume the values are stored in the variables N$, T, and F, respectively. The BASIC statement

```
PRINT N$,T,F
```

would display this information in columns 16 spaces apart from the start of the first value to the start of the second value, and so on. Here, the comma acts as an automatic tabulator. Thus, any line can have "fields" of display starting at column 1, column 17, and column 33. This can be useful when certain types of information are to be displayed. (See, for example, Sections 1.5.1 and 1.5.3.)

If one wished the above information to be *close packed* (printed without any separating spaces), the semicolon would be used in place of the comma. In essence, then, the comma, when *not* used as a punctuation mark, instructs the system to tab 16 spaces before printing; similarly, the semicolon instructs the system to not skip any spaces before printing.

These and other examples of their use will be shown shortly, but for now be aware that the comma and semicolon can have special meanings when not used as punctuation.

1.5 BASIC STATEMENTS FOR THIS CHAPTER

1.5.1 Statement PRINT

Purpose Displays (PRINTs) information at the computer terminal. This information may be text, numeric variable values or string (alphanumeric) variable values (see Section 1.5.3). When text is to be displayed, it must be enclosed in quotation marks (") in the PRINT statement.

Example:
```
PRINT "HELLO. WHAT'S YOUR FIRST NAME"
```

Result of execution:
```
HELLO. WHAT'S YOUR FIRST NAME
```

Example:
```
10 A = 123
20 A$ = "ABC"
30 PRINT A,A$
```

Result of execution:
```
123              ABC
```
←———16 spaces—→

1.5.2 Statement INPUT

Purpose Allows numeric or alphanumeric information to be entered (INPUT) into a BASIC program during its execution. The information is entered through the terminal keyboard and is assigned to a variable specified by the program author. The variable will have the assigned value until changed by another INPUT or LET statement for that variable.

Examples:
```
INPUT N      (for numeric information)
INPUT N$ (for alphanumeric information)
```

Note: Most BASIC systems automatically display a question mark (?) when the INPUT statement is executed. In computer terms, the question mark is called the *input indicator* or *prompt*. Program execution is stopped until the RETURN key is depressed. Also, note that the use of quotes, discussed earlier for string variables in Section 1.4.3, is *not* required when string information is INPUT. The dollar sign instructs the system that *any* input will be assigned as a string variable.

Program example:
```
10 PRINT "HELLO. WHAT'S YOUR FIRST NAME"
20 INPUT N$
30 PRINT N$;" IS A NICE NAME."
40 END
```

Result of execution:
```
HELLO, WHAT'S YOUR FIRST NAME
?SAMMY      (SAMMY is typed and the RETURN key depressed.)
SAMMY IS A NICE NAME.
```

1.5.3 Statement LET

Purpose Assigns values to variables. This action may be "direct," as in LET X = 20 (X would have a value of 20), or it may be "indirect," as in LET X = (2*Y)/3 (X would have a value equal to the result of dividing 3 into the product of 2 times the value of Y). The "*" is the symbol (character) used for multiplication; the "/" is the symbol used for division. It may also be used to assign alphanumeric values, as in LET A$ = "HERE'S THE ANSWER!"

Note: In most BASIC systems, the term LET is optional; so the statement X = 20 is equivalent to LET X = 20. Also, note here that assignment to a string variable requires the use of quotes. The string content *must* be enclosed in quotes.

Example:
```
10 LET NAME$ = "JOHN JONES"
20 LET TEST1 = 100
30 FINALTEST = 89      (Note: LET is omitted.)
40 PRINT "STUDENT","TEST 1", "FINAL"
50 PRINT "-------","------", "-----"
60 PRINT NAME$,TEST1,FINALTEST
70 END
```

Result of execution:
```
STUDENT        TEST 1     FINAL
-------        ------     -----
JOHN JONES     100        89
```

What would happen if the commas in statements 40–60 were replaced by semicolons? (*Note:* Answers to this and other questions found within the text are supplied under their respective chapter and section numbers in Appendix C.)

1.5.4 Statement END

Purpose Ends program execution. On many systems, it is not required. However, in the interest of good programming practice, it should be the last statement in any program.

1.5.5 PROGRAM 1: Years-to-Days Conversion

The statements discussed thus far can be combined to make a program. But what is the program to do? Some stage of program design must be defined that

illustrates the use of these statements. Arbitrarily, then, the program is designed to:

1. Ask for a person's name (PRINT).
2. Store the name entered in a string variable (INPUT).
3. Greet that person with the name entered (PRINT).
4. Skip a line so that the screen is not too crowded (PRINT).
5. Ask for the person's age in years (PRINT).
6. Store the age entered in a numeric variable (INPUT).
7. Convert the age in years to the age in days (LET).
8. Display this age in days (PRINT).
9. End the program (END).

Note: The sample interaction shows a "session" at a microcomputer. It includes creating a NEW program, entering the program statements, SAVEing the program, RUNning the program, LISTing the program, LISTing a single statement (LIST 40), LISTing statements 10 to 30 inclusive (LIST 10,30), and LISTing a nonexistent statement (LIST 120).

All this is shown in the sample as it would appear on an APPLE system monitor, which limits each display line to 40 characters. Any remaining characters on a given line "wrap around" and are shown on the next line displayed. Although this makes absolutely no difference to the APPLE system (a line may contain as many as 255 characters), it is confusing when read by a person. To clarify this, another listing of PROGRAM 1 is shown below the actual session. This and all listings of subsequent programs in this text are in an 80-column format. Be aware that the listings will look different when viewed on the monitor display.

Refer to the listing and run of PROGRAM 1.

```
]NEW

]10 PRINT "HELLO, WHAT'S YOUR FIRST NAME"

]20 INPUT N$

]30 PRINT "HOWDY, ";N$

]40 PRINT

]50 PRINT "TELL ME...WHAT IS YOUR AGE IN YEARS";
```

Statement 10 displays a greeting and asks for user's first name.

Statement 20 automatically displays a "?" and waits for input. Whatever is typed is assigned to N$ when RETURN is depressed.

Statement 30 displays "HOWDY, " (Why the blank space?) and value of N$.

Statement 40 prints a blank line, and statement 50 requests the user's age in years.

```
]60 INPUT A

]70 D=A*365

]80 PRINT "WELL, ";N$;", YOU HAVE BEEN BREATHING"

]90 PRINT "FOR AT LEAST ";D;" DAYS!"

]100 PRINT,"BYE-BYE, ";N$

]110 END

]SAVE PROGRAM 1
]RUN
HELLO. WHAT'S YOUR FIRST NAME
?SAMMY
HOWDY, SAMMY

TELL ME...WHAT IS YOUR AGE IN YEARS?21
WELL, SAMMY, YOU HAVE BEEN BREATHING
FOR AT LEAST 7665 DAYS!
                BYE-BYE, SAMMY
]LIST

10  PRINT "HELLO. WHAT'S YOUR FIR
     ST NAME"
20  INPUT N$
30  PRINT "HOWDY, ";N$
40  PRINT
50  PRINT "TELL ME...WHAT IS YOUR
      AGE IN YEARS";
60  INPUT A
70 D = A * 365
80  PRINT "WELL, ";N$;", YOU HAVE
      BEEN BREATHING"
90  PRINT "FOR AT LEAST ";D;" DAY
     S!"
100  PRINT ,"BYE-BYE, ";N$
110  END

]LIST 40

40  PRINT

]LIST 10,30

10  PRINT "HELLO. WHAT'S YOUR FIR
     ST NAME"
20  INPUT N$
30  PRINT "HOWDY, ";N$
```

Statement 60 automatically displays a "?" and waits until some number is typed and RETURN is depressed. This number is assigned to variable A.

Statement 70 assigns a value to D equal to the value of A times 365 (converting years to days).

Statements 80 and 90 display values of N$ and D, along with appropriate text.

Statement 100 skips 16 spaces and displays a farewell.

Statement 110 ends program execution.

```
]LIST 120

]

]LIST

10  PRINT "HELLO, WHAT'S YOUR FIRST NAME"
20  INPUT N$
30  PRINT "HOWDY, ";N$
40  PRINT
50  PRINT "TELL ME...WHAT IS YOUR AGE IN YEARS";
60  INPUT A
70 D = A * 365
80  PRINT "WELL, ";N$;", YOU HAVE BEEN BREATHING"
90  PRINT "FOR AT LEAST ";D;" DAYS!"
100  PRINT ,"BYE-BYE, ";N$
110  END

]
```

1.6 EDITING BASIC PROGRAMS

Most BASIC systems have some means by which programs may be edited. For example, a PRINT statement with a misspelled word or typographical error may be corrected by editing. Three simple editing techniques are:

1. *Left arrow key* (←). A typographical error may be corrected by backspacing the cursor over the error, entering the correction, and then completing the line being typed. This type of editing can be used only before the RETURN key is depressed for the line being entered.
2. *Retyping the statement.* A statement may be replaced by simply retyping the line number followed by the correct statement.
3. *Deleting lines.* A statement may be deleted entirely by typing the line number only and then depressing the RETURN key. Several lines may be deleted by typing DEL, followed by the beginning and ending line numbers to be deleted. For example, the command:

   ```
   DEL 20,50
   ```

   would delete line numbers 20–50, inclusive.

Although these are only three simple techniques for editing, they will get you started and can be extremely useful. As you become more proficient and at ease with the system, you should become familiar with the more advanced editing techniques described in Appendix B (section B.9, page 221).

1.7 POSERS AND PROBLEMS

(*Note:* Many of the "Posers and Problems" given in this book may be entered and run as programs. Where possible, this should be done, since it will be of help in arriving at the solutions. As a last resort, or to check your work, refer to Appendix C. A ★ indicates a more difficult problem.)

1. Correct any errors found in the following BASIC statements:

```
10 PRIMT "HELLO
20 PRIMT WHAT'S YOUR HEIGHT IN INCHES"
30 INPUT
40 M = 2.54 *
50 PRINT YOU ARE M CENTIMETERS TALL!
60 FINISH
```

2. What is the value of X in each of the following if Y = 6?

```
X = 25
X = (2*Y)/3
X = Y
X = (2*Y)/(3*Y)
X = (Y*Y)/(Y*2)
```

3. What is the purpose of the semicolon in statements 30, 80, and 90 of PROGRAM 1? What would substituting a comma for the semicolon produce? What is the purpose of the comma in statement 100?

4. Note the different positions of the two question marks in the sample RUN of PROGRAM 1. What caused the difference? (*Hint:* Carefully examine statements 10 and 50.)

5. Modify PROGRAM 1 to output the user's age in "heartbeats" (use H as the variable), assuming a pulse rate of 72 beats per minute (and 60 minutes per hour, 24 hours per day).

6. What would result if the following statements were executed?

```
10 A$ = "NAME"
20 B$ = "SCORE"
30 C$ = "AVERAGE"
40 PRINT A$,B$,C$
50 END
```

7. What would result if the following statements were executed? (Assume you input your own name and weight.)

```
10 PRINT "FIRST NAME";
20 INPUT N$
30 PRINT "WEIGHT IN POUNDS";
40 INPUT P
50 K = P/2.2
60 Z = P * 16
70 PRINT,"WOW, ";N$;"!"
80 PRINT "THAT'S ONLY ";K;" KILOGRAMS, BUT, GEE,"
90 PRINT,"IT'S ";Z;" OUNCES!"
100 END
```

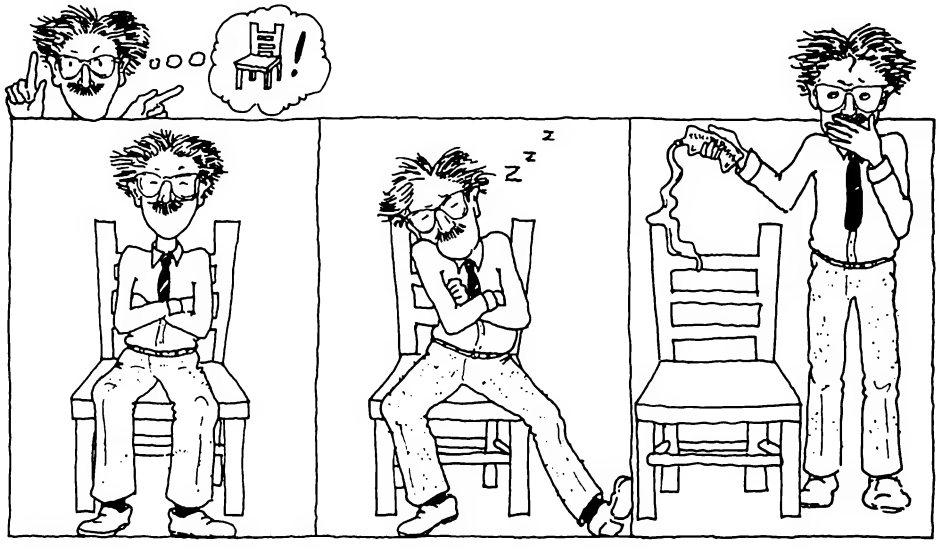
★ 8. Write a program that converts a temperature in Celsius to a temperature in Fahrenheit. The user should enter the temperature for conversion from the keyboard. *Hint:* The formula for conversion is F = (C * 9/5) + 32.

★ 9. Write a program that converts two variables, cups and ounces, into ounces. For example, 2 cups and 3 ounces equal 19 ounces.

★ 10. Write a program that inputs two string variables, first name and last name, and prints out a salutation of your choice using the person's full name.

"Even if you're on the right track, you'll get run over if you just sit there."
—Will Rogers

"Man's mind stretched to a new idea never goes back to its original dimensions."
—Oliver Wendell Holmes



Think About This (for Fun)

What do you sit on, sleep on, and brush your teeth with?

Think About This (Seriously)

Can computer programs teach?

Chapter 2

Now Tell It Where To Go and What To Do with It

2.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Define the purpose and give at least one example of each of the BASIC statements REM, GOTO, IF-THEN, and ON-GOTO (Sections 2.2.1–2.2.4).
2. Define the purpose and give at least one example of each of the BASIC functions RND(1) and INT (Section 2.3).
3. Define the purpose of the BASIC commands LOAD and DELETE (Section 2.4).
4. Alone and unafraid, boot up a microcomputer system (Appendix A).
5. Design, enter, and RUN a BASIC program that includes the statements discussed in Chapter 1.

2.2 BASIC STATEMENTS FOR THIS CHAPTER

2.2.1 Statement REM

Purpose Used as a REMinder or REMark to document the listing of BASIC programs. That is, REM gives a means by which internal notes may be made in the program listing. These notes will provide information about the program, such as which variables are used and their purpose (commonly called a *dictionary of variables*), and will identify special program routines or strategies, separating the program into segments so that the program listing is easy to read. The REM statement is not executed during a program run; thus, the only time these are displayed is after a LIST command.

Example:
```
REM THE VARIABLE 'A' IS THE AGE IN YEARS
```

2.2.2 Statement GOTO

Purpose *Unconditionally* transfers program execution to the specified statement number.

Example:
```
GOTO 100
```

2.2.3 Statement IF-THEN

Purpose *Conditionally* transfers program execution to the specified statement number if, and only if, the defined variable relationship is true.

Examples:
```
IF X = 1 THEN 100
```

(Transfer to statement 100 will occur if X is equal to 1.)

```
IF Y <> Z THEN 100
```

(Transfer to statement 100 will occur if the value of Y is not equal to the value of Z.)

```
IF A <= 2 THEN 100
```

(Transfer to statement 100 will occur if the value of A is less than or equal to 2.)

```
IF A >= 2 THEN 100
```

(Transfer to statement 100 will occur if the value of A is greater than or equal to 2.)

```
IF A$ = "YES" THEN 100
```

(Transfer to statement 100 will occur if the value of A$ is equal to the character string YES.)

2.2.4 Statement ON-GOTO

Purpose Transfers program execution to a specified statement number based on the truncated value of a variable or numerical relationship.

Example:
```
ON X GOTO 100,300,600
```

(Transfer to statement 100 will occur if the truncated value of X is 1; transfer to statement 300 will occur if this value is 2; transfer to statement 600 will occur if this value is 3. If X is less than 1 or greater than 3 in the example above, execution continues with the first statement following the ON-GOTO.)

This example of the ON-GOTO is equivalent to the following three IF-THEN statements:

```
IF X = 1 THEN 100
IF X = 2 THEN 300
IF X = 3 THEN 600
```

By using the ON-GOTO, the same instructions can be given to the system by just one statement:

```
ON X GOTO 100,300,600
```

Note: Truncate is a computer term that means "reduce a number with a decimal fraction to its whole-number value." The truncated values of 3.0001 and 3.9999 are both equal to 3. The truncated values of -3.0001 and -3.9999 are both equal to -4 (since the decimal number is *reduced*).

2.3 SOME VERY BASIC FUNCTIONS

Functions in BASIC are essentially mathematical routines that either come with the computer system (as a *library* of routines or functions) or are defined by the user. Once a function has been defined, it may be used over and over again without the bother of writing out the entire routine.

Two of the most common library functions that are used in instructional computing applications are RND(1) and INT. When executed, the RND(1) function automatically gives some random numeric value between 0.0 and 0.99999999. The INT function truncates any number with a decimal fraction (called a *real* number) to a whole number (called an *integer*).

By using a combination of these functions in BASIC statements, it is possible to generate random numbers within any range desired. This may be used to generate different values for questions containing numbers, randomly selecting questions by number from a "bank" of questions, randomly branching to specified line numbers using ON-GOTO statements, and so on. The following illustration shows how this combination may be used to generate numbers in the range of 1–10, inclusive.

Suppose a BASIC statement looked like this:

```
X = INT(10 * RND(1) + 1)
```

and suppose RND(1) comes up with a random value of 0.58. BASIC is set up so that numerical operations enclosed in parentheses are performed first. Thus, the steps the system follows in computing the value of X would be:

1. 10 * 0.58 = 5.8
2. 5.8 + 1 = 6.8
3. INT of 6.8 = 6

Thus, X will have a value of 6 in this example.

What would be the value of X if RND(1) = 0.99999999? What would be the value of X if RND(1) = 0.01? What is the range of random numbers that could result from the statement:

```
X = INT(10 * RND(1) + 3)
```

What would be the statement that would generate random numbers in the range 1.00–100.00, inclusive? (*Hint:* Note the two decimal places. How is an integer value changed to a real value containing two decimal places? *Answer:* By dividing the integer by 100.) What statement would produce random numbers in the range 5–95, inclusive?

2.4 MODIFICATION OF EXISTING PROGRAMS

In Chapter 1, Problem 5 asked the reader to modify PROGRAM 1 to output (PRINT) the number of heartbeats equivalent to a user's age in years, assuming there were 72 beats per minute. To do this, it is necessary to:

1. Retrieve PROGRAM 1 from the disk.
2. Make the modifications.
3. Save the modified version of PROGRAM 1 as PROGRAM 2.

[By saving the modified program as PROGRAM 2, both the old version (PROGRAM 1) and the new version (PROGRAM 2) are on the diskette. If *only* the new version is desired, the same name (PROGRAM 1, in this case) should be used.]

2.4.1 PROGRAM 2: Adding Heartbeats

Recall that PROGRAM 1 was created by first typing NEW to erase any program in memory and then entering each line, statement by statement. RUN was

typed to test the program, and it was SAVEd on the floppy diskette. Once a program has been SAVEd, it may be retrieved for use or modifications by the command LOAD ⟨filename⟩.

If any changes are made that are to be *permanent* in the program, the command SAVE ⟨filename⟩ must be used. This is illustrated by the listing and sample run of PROGRAM 2. If a SAVEd program is no longer needed, it may be deleted by the command DELETE. Thus, the command for erasing any program is DELETE ⟨filename⟩. But be careful! Once a program is deleted, it is gone, gone, gone!

Run from disk and refer to the listing and run of PROGRAM 2.

```
]LOAD PROGRAM 1
]LIST

10  PRINT "HELLO, WHAT'S YOUR FIRST NAME"
20  INPUT N$
30  PRINT "HOWDY, ";N$
40  PRINT
50  PRINT "TELL ME...WHAT IS YOUR AGE IN YEARS";
60  INPUT A
70 D = A * 365
80  PRINT "WELL, ";N$;", YOU HAVE BEEN BREATHING"
90  PRINT "FOR AT LEAST ";D;" DAYS!"
100  PRINT ,"BYE-BYE, ";N$
110  END

]92 REM ===============

]93 REM MODIFICATIONS ADDED BELOW

]94 REM ===============

]96 H=D*24*60*72

]97 PRINT "AND IN HEARTBEATS, THAT'S"

]98 PRINT "ABOUT ";H;" TOTAL THROBS!"

]99 PRINT "     WOW, ";N$;"!"

]70 D=A*365.25

]RUN
HELLO, WHAT'S YOUR FIRST NAME
?SAMMY
HOWDY, SAMMY
```

Statements 92–99 are entered. (Statement 96 converts age in days, D, to heartbeats, H, since there are 24 hours per day, 60 minutes per hour, and 72 heartbeats per minute.)

Statements 97–99 display the value of variable H, along with appropriate text.

Statement 70 is reentered, giving a more accurate value for days per year (365.25 versus 365).

The program is then RUN.

```
TELL ME...WHAT IS YOUR AGE IN YEARS?43.5
WELL, SAMMY, YOU HAVE BEEN BREATHING
FOR AT LEAST 15888.375 DAYS!
AND IN HEARTBEATS, THAT'S
ABOUT 1.64730672E+09 TOTAL THROBS!
     WOW, SAMMY!
                BYE-BYE, SAMMY

]SAVE PROGRAM 2                          The program is SAVEd as PROGRAM 2.
]LIST                                    A new LIST is requested.

10  PRINT "HELLO, WHAT'S YOUR FIRST NAME"
20  INPUT N$
30  PRINT "HOWDY, ";N$
40  PRINT
50  PRINT "TELL ME...WHAT IS YOUR AGE IN YEARS";
60  INPUT A
70 D = A * 365.25
80  PRINT "WELL, ";N$;", YOU HAVE BEEN BREATHING"
90  PRINT "FOR AT LEAST ";D;" DAYS!"
92  REM ===============
93  REM MODIFICATIONS ADDED BELOW
94  REM ===============
96 H = D * 24 * 60 * 72
97  PRINT "AND IN HEARTBEATS, THAT'S"
98  PRINT "ABOUT ";H;" TOTAL THROBS!"
99  PRINT "     WOW, ";N$;"!"
100  PRINT ,"BYE-BYE, ";N$
110  END

]
```

In summary, we have the following commands:

Command	Example	Action
NEW	NEW	Clears memory of statements.
RUN	RUN	Executes statements in memory.
RUN ⟨filename⟩	RUN PROGRAM 1	LOADs program ⟨filename⟩ from the disk and RUNs it.
LOAD ⟨filename⟩	LOAD PROGRAM 1	LOADs program ⟨filename⟩ from the disk to memory.
LIST	LIST	LISTs the entire program.

LIST nn	LIST 10	LISTs line nn.
LIST nn,mm	LIST 10,100	LISTs lines nn–mm, inclusive.
SAVE ⟨filename⟩	SAVE PROGRAM 1	SAVEs current program in memory on the disk as program ⟨filename⟩.
DEL nn,mm	DEL 10,100	DELetes lines nn–mm, inclusive.
DELETE ⟨filename⟩	DELETE PROGRAM 1	DELETEs program ⟨filename⟩ from the disk.

Note: In the RUN of the program, the value of H is expressed as 1.64730672E+09. This is the method in which the system displays a value of 1,647,306,720. It is also the system's way of expressing *scientific notation*, that is, 1.64730672×10^9. This amounts to one billion, six hundred forty-seven million, three hundred and six thousand, seven hundred and twenty heartbeats! Tho' easily broken, 'tis still a powerful muscle!

2.5 INCORPORATING THE NEW STATEMENTS

The content design of any BASIC program is at the discretion of its author (programmer). The program can be as simple or as complex as the author desires. For example, BASIC may be used in trivial Fahrenheit-to-Celsius temperature conversions or in sophisticated modeling of population dynamics. The point is that a program does *only* what an author has designed it to do—nothing more or less. However, for any program, regardless of its simplicity or complexity, the author must first outline the design and "flow" of the program. On that note, the following program (PROGRAM 3) is designed only to illustrate a use of the statements discussed in this chapter.

2.5.1 PROGRAM 3: Appropriate Responses

The program will ask a question and give only one chance for a correct answer. "Appropriate" responses will be made for either a correct or incorrect answer. The program will then ask a final question related to age. The user will be informed if the answer is too low or too high. For answers that are too high, an additional comment will be randomly selected from three choices. The question will be repeated until the correct answer is given.

This is a general outline of what the program is designed to do. The BASIC statements needed to accomplish this are shown in the program listing. (Now, try to relax when you see the "long" listing of the program. Think about what each statement instructs the system to do and mentally follow its execution.)

RUN from disk and refer to the listing and run of PROGRAM 3.

```
]LOAD PROGRAM 3
]LIST

10  REM       PROGRAM 3
20  REM  ================
30  REM  ASK SOME QUESTION
40  REM  GET AN ANSWER AND
50  REM  CHECK FOR ACCURACY
60  REM  ================
70  PRINT "WHAT STATE FOLLOWS ALASKA"
80  PRINT "IN TOTAL LAND AREA";
90  INPUT R$
100  IF R$ = "TEXAS" THEN 130
110  PRINT ,"NOPE, IT'S TEXAS!"
120  GOTO 140
130  PRINT  TAB( 3);"YEEE-HAAA!  YOUR ANSWER
     IS CORRECT!"
140  PRINT
150  REM  ================
160  REM  ASK ANOTHER QUESTION
170  REM  AND CHECK FOR HIGH OR
180  REM  LOW ANSWER INPUT
190  REM  ================
200  PRINT "WHAT WAS THE PERPETUAL AGE"
210  PRINT "OF THE LATE JACK BENNY";
220  INPUT R
230  IF R < 39 THEN 290
240  IF R > 39 THEN 320
250  REM  ===THEN INPUT EQUALS 39===
260  PRINT
270  PRINT "DIDN'T LOOK IT, DID HE..."
280  GOTO 490
290  PRINT ,"TOO LOW..."
300  REM  ===REPEAT THE QUESTION===
310  GOTO 140
320  PRINT ,"TOO HIGH..."
330  REM  ================
340  REM  GET A RANDOM COMMENT FOR
350  REM  ANY ANSWER THAT IS HIGH
360  REM  THEN REPEAT THE QUESTION
370  REM  ================
380 X =  INT (3 * RND (1) + 1)
390  ON X GOTO 400,420,440
```

Statements 70 and 80 (the first executed statements) print the question.

Statement 90 displays a question mark (note where it is displayed!), waits for input, and stores it in the string variable R$. (*Note:* From now on, reference to statements will sometimes be made by number only. That is, statement 100, for example, will simply be referred to as 100.)

100 checks for value of R$ equal to TEXAS. If so, transfer to 130 occurs. If not, 110 is executed, and then 120 causes transfer to 140 (skipping 130, the response for the correct answer).

140 prints a blank line.

200 and 210 print the next question.

220 displays a "?"; waits for input; and stores it in variable R.

230 checks for R less than 39. If so, transfer is to 290. If not, execution continues to 240, which checks for R greater than 39. If so, transfer is to 320. If not, 260 and 270 are executed. (230–270 essentially say that if R is neither greater nor less than 39, then it must be equal to it.)

290 (from 230, if R< 39) lets user know they are too low, and 310 returns to the question (140).

320 (from 240, if R>39) lets user know they are too high.

380 gives a random value for X between 3 and 1, inclusive.

390 transfers execution to 400 if X = 1, to 420 if X = 2, or to 440 if X = 3. These statements print an additional comment and return the execution to 140 (by 410, 430, or 450).

```
400  PRINT "NOW THAT IS OLD!"
410  GOTO 140
420  PRINT "ARE YOU TRYING TO BE CRUEL?"
430  GOTO 140
440  PRINT "HAVE YOU NO SYMPATHY?"
450  GOTO 140
460  REM  ===============
470  REM  END THE PROGRAM
480  REM  ===============
490  PRINT
500  PRINT
510  PRINT ,"BYE-BYE, FRIENDS..."
520  END


]RUN
WHAT STATE FOLLOWS ALASKA
IN TOTAL LAND AREA?CALIFORNIA
                NOPE, IT'S TEXAS!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?33
                TOO LOW...

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?69
                TOO HIGH...
HAVE YOU NO SYMPATHY?

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?66
                TOO HIGH...
NOW THAT IS OLD!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?55
                TOO HIGH...
NOW THAT IS OLD!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?39

DIDN'T LOOK IT, DID HE...

                BYE-BYE, FRIENDS...
```

If R is neither less than nor greater than 39 (see statements 230–270), 280 transfers execution to 490 which prints a blank line, as does 500. A farewell is printed by 510 and the program ends at 520.

```
]RUN
WHAT STATE FOLLOWS ALASKA
IN TOTAL LAND AREA?TEXAS
  YEEE-HAAA!  YOUR ANSWER IS CORRECT!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?39

DIDN'T LOOK IT, DID HE...

                BYE-BYE, FRIENDS...

]
```

2.6 POSERS AND PROBLEMS

1. What is the difference between the variables R and R$ in PROGRAM 3?

2. What would be the result if statement 390 in PROGRAM 3 read ON X GOTO 440,400,420?

3. Modify statements 400, 420, and 440 in PROGRAM 3 to give comments of your choosing.

4. What should be done to PROGRAM 3 so that it would ask for your age instead of Jack Benny's?

5. What should be done to PROGRAM 3 in order to select a random comment from five choices instead of three?

6. What changes should be made to PROGRAM 3 in order to ask for the third largest state by land area instead of the second?

7. How should PROGRAM 3 be modified to ask for the user's first name at the start of the program and then refer to them by name when "BYE-BYE..." is executed in statement 510?

8. Add some REM statements to PROGRAM 1 so that the variables are made clearer to someone looking at the listing of the program for the first time.

9. Write a statement that will randomly give a value for variable X that is between 100 and 25, inclusive.

10. What is the range of numbers that could randomly be generated by the statement:

```
X = INT(25 * RND(1) + 5)
```

★ *11.* Write a program that asks for the user's height in inches and then prints "TALL" if the user is over six feet, "SHORT" if under five feet, or "AVERAGE" if between five and six feet, inclusive.

★ *12.* Write a program that inputs the lengths of the sides of a triangle as variables A, B, and C (largest side last) and determines if it is a right triangle. [*Hint*: For right triangles, C ^ 2 = A ^ 2 + B ^ 2. The caret (^) is the Apple's way to "raise to the power of.")]

★ *13.* Write a program that inputs a number and prints "THREE" if it is a 3, "SIX" if it is a 6, "NINE" if it is a 9, or "NEITHER 3, 6, nor 9" if it is not equal to either 3, 6, or 9.

"Don't put off for tomorrow what you can do today, because if you enjoy it today you can do it again tomorrow."
—James A. Michener

Think About This (for Fun)

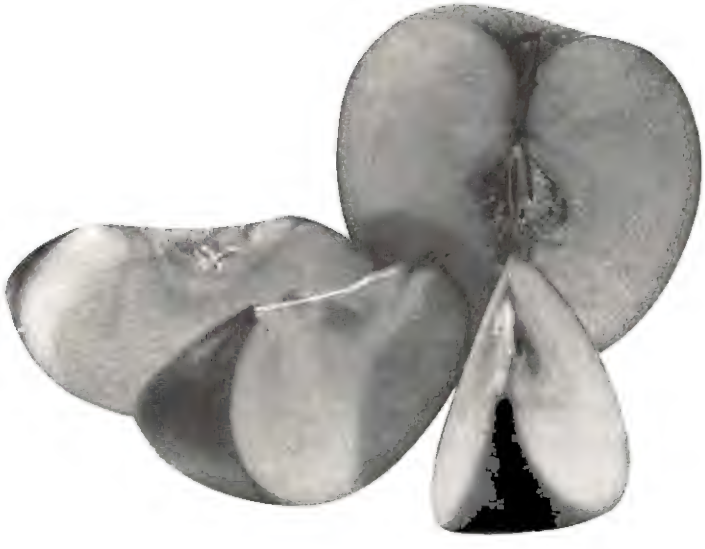
What is the exact opposite of *not in*?

Think About This (Seriously)

Is the use of computers in instruction just another "educational fad"?

Chapter

3



Take a Ride on the Loop-D-Loop

3.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Define and give at least one example of each of the BASIC statements HOME, DATA-READ, RESTORE, and FOR-NEXT (Sections 3.2.1–3.2.4).
2. Enter and RUN each of the BASIC programs used as statement examples in this chapter.

3.2 BASIC STATEMENTS FOR THIS CHAPTER

3.2.1 Statement HOME

Purpose HOME clears (erases) all display and "homes" the cursor in the upper left corner of the TV or monitor screen. This use is particularly appropriate in instructional computing since it allows information, examples, questions, and so on, to be displayed in a frame-by-frame fashion.

Example:
```
HOME
```

3.2.2 Statement Pair DATA-READ

Purpose DATA allows information (numeric or string) to be stored in a program for use at various stages throughout its execution. The pieces of information are generally referred to as *data elements*, with each element separated (*delimited*) by a comma. READ assigns the defined value of a data element to a specified variable and "moves" a data "marker" or "pointer" to the next data element. The data type (numeric or string) *must* match the variable type (numeric or string). For example, "ABC" cannot be assigned to a numeric variable.

There are two important notes to be made in regard to the elements in the DATA statements:

1. *Never* place a comma at the end of the DATA statement. The system may take the space character following the comma as the next data element!
2. On the Apple system it is not actually necessary to enclose string elements in quotation marks. However, there are certain sequences of characters that the system may misinterpret if the string is not enclosed in quotes. Therefore, as a safeguard, always enclose string DATA elements with quotation marks. It is worth the extra keystrokes required to do this in order to avoid any potential difficulties that may otherwise result.

Example:
```
10 DATA 1,"ABC",2
20 READ N
30 PRINT N
40 READ N$
50 PRINT N$
60 READ N
70 PRINT N
80 END
```

Why did the value of the variable N change? If statement

```
75 PRINT N$
```

were added, what would be the result of running the program again?

Example:
```
10 DATA 1,"ABC",2,"DEF"
20 READ N,N$,N1
30 PRINT N,N$,N1
40 READ N$
50 PRINT N1;N;N$
60 END
```

Why did the value of the variable N$ change? What caused the display format of PRINT statements 30 and 50 to be different?

(After mentally tracing the program execution, enter and RUN the above examples to check your mental interpretations.)

3.2.3 Statement RESTORE

Purpose Moves the data pointer to the first data element in the first DATA statement.

Example:
```
10 X = 0
20 DATA 1,"ABC",2,"DEF"
30 READ N,N$,N1,N1$
40 PRINT N,N$,N1,N1$
50 IF X = 2 THEN 100
60 X = X + 1
70 PRINT "THE VALUE OF X IS ";X
90 GOTO 30
100 END
```

Enter and RUN the above program and note what happens. Add the statement 80 RESTORE and RUN again. Note the results.

What is the position of the data pointer after statement 30 has been executed but *before* the RESTORE statement is added? What caused the error message in the first RUN? What caused the program to stop execution after the statement 80 RESTORE was added? What would the program do if statements 50–70 were deleted (after 80 RESTORE was included in the program)? Think this through before RUNning; otherwise, remember that simultaneously depressing the CONTROL and C keys (CTRL-C) will halt the execution of a "runaway" program!

Note: On many BASIC systems, statements such as 10 X = 0 (as in the program above, for example) are not needed because all variables are automatically "initialized" (set) to zero. However, it is good programming practice to initialize variables to zero in any program.

3.2.4 Statement Pair FOR-NEXT

Purpose Defines the number of times (loops) a series of consecutive BASIC statements are to be repeated. FOR defines the variable used as a counter for the repeats and the lower and upper limits of the count. NEXT increases the variable count by 1 (or the defined STEP size) and checks to see if the upper limit of the FOR is exceeded. If not, execution is transferred to the statement

immediately following the FOR statement. If the upper limit is exceeded, execution is transferred to the statement immediately following the NEXT statement.

The variable names in the loop defined by a FOR-NEXT must be identical. Note also that the start and/or limit of the loop may be defined by variable names, as in FOR X = Y TO Z.

Examples:

```
10 FOR X = 1 TO 10
20 PRINT X,X * X
30 NEXT X
40 PRINT,"THAT'S ALL..."
50 END

10 FOR X = 1 TO 10 STEP 2
20 PRINT X,X*X*X
30 PRINT,"MORE TO COME..."
40 NEXT X
50 PRINT,"THAT'S ALL..."
60 END

10 A = 10
20 B = -10
30 FOR C = A TO B STEP -2
40 PRINT C;" ";
50 NEXT C
60 END
```

(Mentally trace the execution, and then enter and RUN each program.)

3.3 INCORPORATING THE NEW STATEMENTS

With the addition of the statements in this chapter, a BASIC program may be designed that provides more of a utility than the earlier programs. One of the many uses of computer programs involves searching a list of information (commonly called a *data base*) for key elements that may be specified by a user. For example, data bases may be searched for financial accounts that are overdue by 30, 60, or 90 days; address lists may be searched for ZIP codes; employee rolls may be searched for persons who have special deductions; and so on. The following program illustrates one search technique.

3.3.1 PROGRAM 4: Searching for a Range of Values

The program contains a list of DATA elements representing pairs of hypothetical names and scores. This list is to be searched for scores that fall within a specified maximum and minimum range. A list of names and scores that are within this range is printed. Following this, the user is given an option to do another search.

What will this require as the program is mentally designed? In outline form, there must be at least:

1. Prompts to get the maximum and minimum scores (PRINTs).
2. Entry of these scores (INPUTs).
3. A loop (FOR) to:
   a. READ the DATA.
   b. Check (IF-THENs) to see if the current score read is the last data element in the list and, if not, to see if it is in the maximum-minimum range.
   c. Display (PRINT) the name and score if in the range.
   d. Continue the search (NEXT).
4. A prompt for another search option (PRINT and INPUT).
5. Movement of the data pointer back to the first data element if another search is to be done (RESTORE), followed by repetition of the total process (GOTO).
6. A list of names and scores in the program (DATA).
7. An end to the program (PRINT and END).

RUN from disk and refer to the listing and run of PROGRAM 4.

```
]LOAD PROGRAM 4
]LIST

10  REM  ===============
20  REM  PROGRAM 4 DESCRIPTION
30  REM  ===============
40  REM  KEY SEARCH OF 50 OR LESS DATA ELEMENT
         PAIRS.
50  REM  PROGRAM SEARCHES FOR A MINIMUM-MAXIMUM
         RANGE OF SCORES.
60  REM  DATA ELEMENTS ARE IN SEQUENCE: NAME,
         SCORE.
70  REM  LAST SEQUENCE OF DATA ELEMENTS: "X",0
80  REM  ===============
90  REM  VARIABLE DICTIONARY
100  REM  ===============
110  REM  N$ - HYPOTHETICAL NAME
120  REM  S - HYPOTHETICAL SCORE
130  REM  M1 - MAXIMUM SCORE
140  REM  M2 - MINIMUM SCORE
150  REM  I - LOOP COUNTER
160  REM  F - "FLAG" FOR FINDING AT LEAST ONE
          MATCH
```

Statements 10–200 give a brief *documentation* of the program, describing its primary functions and listing important variables and their use.

```
170  REM  ===============
180  REM  SET THE MATCH FLAG TO ZERO, CLEAR
190  REM  THE SCREEN, AND GET THE RANGE SOUGHT.
200  REM  ===============
210 F = 0
220  HOME
230  PRINT "MAXIMUM SCORE";
240  INPUT M1
250  PRINT "MINIMUM SCORE";
260  INPUT M2
270  PRINT "NAMES WITH SCORES IN THE RANGE:
     ";M2;"-";M1
280  PRINT "NAME","SCORE"
290  PRINT "----","-----"
300  REM  ===============
310  REM  DO THE SEARCH LOOP
320  REM  ===============
330  FOR I = 1 TO 50
340  READ N$,S
350  REM  ===END OF DATA LIST?===
360  IF N$ = "X" THEN 460
370  IF S > M1 THEN 420
380  IF S < M2 THEN 420
390  PRINT N$,S
400  REM  ===FLAG A SEARCH MATCH===
410 F = 1
420  NEXT I
430  REM  ===============
440  REM  END OF CURRENT SEARCH
450  REM  ===============
460  IF F = 1 THEN 480
470  PRINT "***NONE FOUND***"
480  PRINT "DO YOU WISH ANOTHER SEARCH (Y OR N)";
490  INPUT Z$
500  IF Z$ <  > "Y" THEN 1000
510  RESTORE
520  GOTO 210
530  REM  ===============
540  REM       DATA LIST
550  REM  ===============
560  DATA "SUE",67,"BOB",55,"JACK",98,"MARY",99,
     "STAN",50,"ROB",72
570  DATA "LETA",77,"ALEX",66,"SUSAN",85,"MARIA"
     ,99,"FRAN",70
580  DATA "BOBBIE",100,"CHARLES",64,"BILLY",66,
     "MAGGIE",86
590  DATA  "DONNA",91,"YANCY",77,"TRACY",89,
     "KAREN",100,"BUCK",90
600  REM  ===ROOM FOR MORE DATA===
999  DATA  "X",0
```

210 initializes variable F to zero. (F is used as a "flag" to indicate if a search found any match: If a match is found, F is set to 1; otherwise, F remains 0.)

220 clears the screen and HOMEs the cursor at line 1, column 1 on the screen.

230–260 obtain maximum and minimum range of scores desired by user.

270–290 PRINT a heading for the list.

330–420 define a FOR-NEXT loop.

330 assigns I as loop counter, sets its intial value to 1, and defines its upper limit as 50 (i.e., the loop will execute maximum 50 times).

340 READs a DATA element pair from data list beginning at 560 and assigns values read to N$ and S, respectively. (The first time READ is executed, N$ has a value of SUE, S equals 67. Data pointer then moves to BOB in preparation for next READ.)

360 checks value of N$. If equal to the character X, execution transfers to 460. (X is arbitrarily defined as last "name" in data list. This lets program know when last data element pair has been read.)

370–380 check S, the score just READ, to see if outside max-min range (greater than M1 or less than M2). If so, execution transfers to 420 where loop repeats if value (I + 1) does not exceed defined upper limit of 50.

If S falls within range (neither greater than M1 nor less than M2), 390 PRINTs current values of N$ and S.

410 sets flag F to 1, indicating a match found.

```
1000  PRINT "*** SEARCH COMPLETED ***"
1010  END

]LOAD PROGRAM 4
]RUN
```

[Clear screen]

```
MAXIMUM SCORE?100
MINIMUM SCORE?90
NAMES WITH SCORES IN THE RANGE: 90-100
NAME                SCORE
----                -----
JACK                98
MARY                99
MARIA               99
BOBBIE              100
DONNA               91
KAREN               100
BUCK                90
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
MAXIMUM SCORE?89
MINIMUM SCORE?80
NAMES WITH SCORES IN THE RANGE: 80-89
NAME                SCORE
----                -----
SUSAN               85
MAGGIE              86
TRACY               89
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
MAXIMUM SCORE?79
MINIMUM SCORE?70
NAMES WITH SCORES IN THE RANGE: 70-79
NAME                SCORE
----                -----
ROB                 72
LETA                77
FRAN                70
YANCY               77
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
MAXIMUM SCORE?69
MINIMUM SCORE?60
```

420 repeats loop if not yet occurred 50 times.

460 can be executed from one of two sources: 360, if last data element has been READ; or 420, if NEXT I exceeds limit defined in statement 330.

460 checks value of F. If F = 1 (indicating a match), transfer is to 480. If F≠ 1, no match was found. 470 then PRINTs that message.

480–500 give option for another search. If INPUT value for Z$ not equal to Y (for Yes), transfer is to 1000 and program ENDs. Otherwise, 510 RESTOREs data pointer to first data element, and 520 transfers execution back to 210 for another search.

```
NAMES WITH SCORES IN THE RANGE: 60-69
NAME                 SCORE
----                 -----
SUE                  67
ALEX                 66
CHARLES              64
BILLY                66
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
MAXIMUM SCORE?59
MINIMUM SCORE?50
NAMES WITH SCORES IN THE RANGE: 50-59
NAME                 SCORE
----                 -----
BOB                  55
STAN                 50
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
MAXIMUM SCORE?49
MINIMUM SCORE?40
NAMES WITH SCORES IN THE RANGE: 40-49
NAME                 SCORE
----                 -----
***NONE FOUND***
DO YOU WISH ANOTHER SEARCH (Y OR N)?N
*** SEARCH COMPLETED ***
```

3.3.2 Additional Comments on PROGRAM 4

DATA statements may be included anywhere in a BASIC program. They are not executed as, for example, a PRINT or INPUT statement would be. Their only use is to contain data (information) that is to be READ and assigned to variables. The DATA statements in PROGRAM 4 are placed near the end of the program so that additional DATA statements may be added if desired. The last data element pair, "X" and 0, is given the highest number possible for a DATA statement (999 in this case). Thus, additional DATA statements could be inserted between statements 600–999 if there were a need to add more data.

3.4 A TIME-SAVING TECHNIQUE

There may be times when the reader wishes to SAVE both the "old" and "new" versions of a program. The "new" (modified) version of a program may be SAVEd by simply giving it a new (unique) name when the SAVE command is

issued. We did this earlier in Chapter 2 when PROGRAM 1 was modified (with the heartbeats) and SAVEd as PROGRAM 2.

To illustrate this technique, PROGRAM 4 will be modified, renamed, and SAVEd as PROGRAM 5. This means that both the old (PROGRAM 4) and the new (PROGRAM 5) programs will be available for future use.

3.4.1 PROGRAM 5: Searching for a Specific Value

Arbitrarily, this new version of PROGRAM 4 will perform a search for only one user-specified score, listing all the names with that score.

First, the LIST of PROGRAM 4 is studied. Examination of the list of statements shows which lines need to be modified. These changes are then made, as indicated in the session that follows.

The program is then RUN to test these modifications. On completion of a successful RUN, the program is SAVEd as PROGRAM 5. It is then listed for examination. Note that a new program (PROGRAM 5), based on another program (PROGRAM 4), has been created and SAVEd without the time and trouble required to completely type the new version.

Run from disk and refer to the listing and run of PROGRAM 5.

```
]LOAD PROGRAM 4
]20  REM  PROGRAM 5 DESCRIPTION

]50  REM  PROGRAM SEARCHES FOR A SPECIFIED
     SCORE.
]130  REM  M1 - SCORE SOUGHT

]140

]160  REM  F - COUNTER FOR THE NUMBER OF MATCHES
      FOUND
]230 PRINT "SCORE SOUGHT";

]DEL 250,260

]270 PRINT "NAMES WITH A SCORE OF ";M1

]370 IF S <> M1 THEN 420

]380

]400  REM  ===COUNT THE NUMBER OF MATCHES
      FOUND===
]410 F = F + 1

]460 PRINT "THIS SEARCH FOUND ";F;" MATCH(ES)"

]470 PRINT
```

20 and 50 retyped in program description.

130 retyped to reflect program change.

140 is deleted (no longer needed).

160 retyped to reflect program change.

230 retyped to reflect program change.

250 and 260 deleted (not needed).

270 retyped to reflect program change.

370 retyped to perform check for match of score sought.

380 is deleted (not needed).

400 retyped to reflect program change.

410 changed from flag to counter of matches found.

460 and 470 changed to show number of matches found.

```
]RUN
```

[Clear screen]

```
SCORE SOUGHT?100
NAMES WITH A SCORE OF 100
NAME                 SCORE
----                 -----
BOBBIE               100
KAREN                100
THIS SEARCH FOUND 2 MATCH(ES)

DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
SCORE SOUGHT?75
NAMES WITH A SCORE OF 75
NAME                 SCORE
----                 -----
THIS SEARCH FOUND 0 MATCH(ES)

DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
```

[Clear screen]

```
SCORE SOUGHT?50
NAMES WITH A SCORE OF 50
NAME                 SCORE
----                 -----
STAN                 50
THIS SEARCH FOUND 1 MATCH(ES)

DO YOU WISH ANOTHER SEARCH (Y OR N)?N
*** SEARCH COMPLETED ***

]SAVE PROGRAM 5

]LIST

10  REM  ===============
20  REM  PROGRAM 5 DESCRIPTION
30  REM  ===============
40  REM  KEY SEARCH OF 50 OR LESS DATA ELEMENT
         PAIRS.
50  REM  PROGRAM SEARCHES FOR A SPECIFIED SCORE.
60  REM  DATA ELEMENTS ARE IN SEQUENCE: NAME,
         SCORE.
70  REM  LAST SEQUENCE OF DATA ELEMENTS IS "X",0
```

```
80  REM  ===============
90  REM  VARIABLE DICTIONARY
100  REM  ===============
110  REM  N$ - HYPOTHETICAL NAME
120  REM  S - HYPOTHETICAL SCORE
130  REM  M1 - SCORE SOUGHT
150  REM  I - LOOP COUNTER
160  REM  F - COUNTER FOR THE NUMBER OF MATCHES
     FOUND
170  REM  ===============
180  REM  SET THE MATCH FLAG TO ZERO, CLEAR
190  REM  THE SCREEN, AND GET THE RANGE SOUGHT.
200  REM  ===============
210 F = 0
220  HOME
230  PRINT "SCORE SOUGHT";
240  INPUT M1
270  PRINT "NAMES WITH A SCORE OF ";M1
280  PRINT "NAME","SCORE"
290  PRINT "----","-----"
300  REM  ===============
310  REM  DO THE SEARCH LOOP
320  REM  ===============
330  FOR I = 1 TO 50
340  READ N$,S
350  REM  ===END OF DATA LIST?===
360  IF N$ = "X" THEN 460
370  IF S <  > M1 THEN 420
390  PRINT N$,S
400  REM   ===COUNT THE NUMBER OF MATCHES
     FOUND===
410 F = F + 1
420  NEXT I
430  REM  ===============
440  REM  END OF CURRENT SEARCH
450  REM  ===============
460  PRINT "THIS SEARCH FOUND ";F;" MATCH(ES)"
470  PRINT
480  PRINT "DO YOU WISH ANOTHER SEARCH (Y OR N)";
490  INPUT Z$
500  IF Z$ <  > "Y" THEN 1000
510  RESTORE
520  GOTO 210
530  REM  ===============
540  REM       DATA LIST
550  REM  ===============
560  DATA   "SUE",67,"BOB",55,"JACK",98,"MARY"
     ,99,"STAN",50,"ROB",72
570  DATA   "LETA",77,"ALEX",66,"SUSAN",85,
     "MARIA",99,"FRAN",70
```

```
 580  DATA   "BOBBIE",100,"CHARLES",64,"BILLY"
      ,66,"MAGGIE",86
 590  DATA   "DONNA",91,"YANCY",77,"TRACY",89,
      "KAREN",100,"BUCK",90
 600  REM  ===ROOM FOR MORE DATA===
 999  DATA "X",0
1000  PRINT "*** SEARCH COMPLETED ***"
1010  END
```

3.5 POSERS AND PROBLEMS

1. Correct any errors (if, in fact, there are any) in the following three programs:

```
10 FOR X = 1 TO 10
20 PRINT Y,Y*Y
30 NEXT Y
40 END
```

```
10 DATA 1,"ABC",2
20 READ N,N$,N1$
30 PRINT N,N$,N1$
40 END
```

```
10 DATA 4,5,6
20 FOR X = 1 TO 3
30 READ A
40 PRINT A
50 NEXT X
60 READ A
70 PRINT A
80 END
```

2. Below are some student data for a name and test score. Complete the program so that the name and score are printed in columnar form.

```
10 DATA "CHUCK",95,"MARY",80,"PHIL",95,"JEANNIE",35
20 FOR I = 1 TO 4
30 READ S$,S
??
```

3. Modify your program in Problem 2 to print the average of the scores after printing the list of names and scores.

4. Write a search program in which a list of data elements consists of hypothetical names, hair color, eye color, and height in inches. A list is to be printed that shows the above information based upon a search of a user-specified eye color. For example, if BLUE is input in response to

"EYE COLOR?", the name, hair color, eye color, and height in inches for all blue-eyed people would be printed. The program should also give the option to conduct another search.

Note: This program may be easily completed by modifying PROGRAM 5. For example, PRINT statements describing the program and prompting for the eye color will have to be modified and/or added. The INPUT will have to be a string variable. The READ statement will need to read four DATA elements. The IF-THEN check will have to compare the INPUT variable and the eye-color variable just read. The PRINT statement showing a match will have to be written so that four variable values are displayed on one line—consider using TAB between the variables in the PRINT statement. Finally, the DATA statements will have to contain four elements (name, hair color, eye color, and height in inches).

5. Enter and RUN the following program. Be prepared to discuss its flow.

```
5 SPEED = 100
10 FOR X = 10 TO 1 STEP -1
20 IF X > 1 THEN 50
30 PRINT X;" LITTLE RABBIT, SEE ITS TAIL!"
40 GOTO 55
50 PRINT X;" LITTLE RABBITS, SEE THEIR TAILS!"
55 PRINT,
60       FOR I = 1 TO X
70       PRINT "* ";
80       NEXT I
90 PRINT
100 NEXT X
110 PRINT
120 PRINT "AND THEN THERE WERE NONE..."
125 SPEED = 255
130 END
```

In particular, what is the purpose of "STEP -1" in statement 10, the comma in statement 55, the semicolon in statement 70, and the PRINT in statement 90? [*Note:* Statements 5 and 125 in the above program introduce SPEED which may be used either as a statement or command. It is used to control the rate of character display on the screen. Its value is relative and for our purposes may vary from 10 (very slow) to 255 (normal rate).]

6. Write a program which converts Celsius to Fahrenheit and PRINTs an equivalence table for every 5 Celsius degrees from 0 to 100, inclusive. *Hint:* F = 32 + (C * 9/5).

7. Write a program that will PRINT the cube of the numbers 1 to 10, inclusive.

"Frustration is not having anyone to blame but yourself."
—Bits and Pieces

"Work spares us from three great evils: boredom, vice and need."
—Voltaire

Think About This (for Fun)

I have two U.S. coins that total $0.55. One of them is *not* a nickel. What are the coins?

Think About This (Seriously)

Could the proliferation of microcomputers in the school *and* home alter the traditional classroom setting as we now know it? If so, how?

Chapter

4

DIM It! There Must Be an Easier Way! Array! Array! There Is!

4.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Define and give at least one example of a subscripted variable (Section 4.2.1).
2. Define and give at least one example of both a one-dimensional and a two-dimensional array (Sections 4.2.1 and 4.2.2).
3. Define and give at least one example of each of the BASIC statements DIM and GOSUB-RETURN (Section 4.4).
4. Design, enter, and RUN a BASIC program of your own choosing that includes a one-dimensional array and the BASIC statements for this chapter.

4.2 ARRAYS

4.2.1 One-Dimensional Arrays

For our purposes, a *one-dimensional array* is just an organized list of information. That information could be any string and/or numeric values: student names, states, chemical names, school districts, test scores, ages, weights, years, and so on.

Recall the four names and test scores given in Problem 2 of Chapter 3. Using one-dimensional arrays in BASIC, we can easily make lists of those names and scores.

Consider the BASIC statements needed. First, there is information (names and scores) that will be used; thus, there will be a need for DATA statements. Of course, if there is DATA, it will need to be READ. Also, since there are two sets of four elements (four names and four scores), a FOR-NEXT loop could be used to do the READing. Thus:

```
10 DATA "CHUCK","MARY","PHIL","JEANNIE"
20 FOR I = 1 TO 4
30 READ N$(I)
40 NEXT I
```

This should appear somewhat familiar, with the exception of statement 30, READ N$(I). N$(I) is an example of another type of variable. This type, however, uses an "internal" variable, (I), to distinguish one value of N$(I) from the others. Remember, the variable I is going to have a value that may be 1, 2, 3, or 4 (FOR I = 1 TO 4). Thus, the variable values in this example are:

```
N$(1) = CHUCK
N$(2) = MARY
N$(3) = PHIL
N$(4) = JEANNIE
```

Or, said another way, there is a 4-item list (one-dimensional array) of N$(I) values:

I	N$(I)
1.	CHUCK
2.	MARY
3.	PHIL
4.	JEANNIE

The name given to these types of variables is *subscripted variables*. The value of N$(1), pronounced "N$ sub 1," is equal to the string CHUCK; the value of N$(2) is equal to the string MARY, and so on.

It is quite simple to build a series of lists using subscripted variables. Consider:

```
10 DATA "CHUCK",95,"MARY",80,"PHIL",95,"JEANNIE",35
20 FOR I = 1 TO 4
30 READ N$(I),S(I)
40 NEXT I
```

If these statements were to be executed, what would be the value of N$(3)? Of S(4)? If the following statements were added to those above, what would be the result of execution?

```
50 FOR I = 4 TO 1 STEP -1
60 PRINT N$(I),S(I)
70 NEXT I
80 END
```

(Mentally trace the execution; then enter and RUN to check your mental interpretation.)

4.2.2 Two-Dimensional Arrays

A one-dimensional array is nothing more than a list of data. A *two-dimensional* array is a table of data in rows and columns.

Assume there are two test scores for each of those students above. A table, consisting of 4 rows and 2 columns, might look like this:

Name	Test 1	Test 2
CHUCK	95	80
MARY	80	82
PHIL	95	93
JEANNIE	35	98

Note: The table in this example is composed of the test scores. The names of the students cannot be included in the table because they are *string* variables, and the two-dimensional array is defined as a *numeric* array. In other words, variable types (string and numeric) cannot be mixed in an array. However, just as it is possible to define a two-dimensional numeric array, a two-dimensional string array may also be defined. Just don't mix them!

How could a two-dimensional array of this information be formed? Again, there is information (names and scores) that will be used; so DATA statements are appropriate. This information in statement form would be:

```
10 DATA "CHUCK",95,80
20 DATA "MARY",80,82
30 DATA "PHIL",95,93
40 DATA "JEANNIE",35,98
```

(The DATA statements above could be combined into one or two statements. However, they are listed as four separate statements for the sake of clarity.)

The difficulty is in determining how the DATA should be READ. Examine the sequence of information: one name, followed by two scores; then the next name, followed by two scores; and so on. Since there are 4 rows (names) and each must be READ, the first loop to be defined will be FOR I = 1 TO 4. However, before the next name is READ, there are 2 scores to be assigned (READ). So, another loop, FOR J = 1 TO 2, needs to be defined. Thus:

```
50 FOR I = 1 TO 4
60 READ N$(I)
70         FOR J = 1 TO 2
80             READ S(I,J)
90         NEXT J
100 NEXT I
```

(*Note:* The indentation is for clarity only.)

The variable I is READing the rows of the table, and the variable J is READing the columns. The J loop is said to be *nested* within the I loop. Examine the table above. What is the value of S(I,J) when I = 1 and J = 1? When I = 3 and J = 2? The important point to remember is that I and J are nothing more than numbers that define a row and column, respectively.

A more productive use of two-dimensional arrays may be illustrated by adding the following statements to the program:

```
45 PRINT "NAME","TEST1","TEST2"
46 PRINT "----","-----","-----"
65 PRINT N$(I),
81 PRINT S(I,J),
82 REM T(I) = CUMULATIVE TOTAL EACH STUDENT'S SCORE
84 T(I) = T(I) + S(I,J)
86 REM T = CUMULATIVE TOTAL OF ALL SCORES
88 T = T + S(I,J)
94 REM SKIP A LINE PRIOR TO PRINT OF NEXT NAME
95 PRINT
110 PRINT
120 FOR I = 1 TO 4
130 PRINT N$(I);"'S AVERAGE IS ";T(I)/2
140 NEXT I
150 PRINT "THE CLASS AVERAGE IS ";T/8
160 END
```

Combine all the statements in this section into one program; then enter and RUN it. What is the purpose of the comma at the end of statements 65 and 81? Why is a blank PRINT needed in statement 95? Examine the LIST of the complete program very carefully and mentally follow the execution of each statement.

4.3 EXAMPLES OF THE USE OF ONE-DIMENSIONAL ARRAYS

There are many more applications of one- and two-dimensional arrays in instructional computing than just building lists or tables of names and scores. The following two programs give examples of some of these uses.

4.3.1 PROGRAM 6: Searching Based on Optional Keys

One use of one-dimensional arrays may be seen in PROGRAM 6, an expanded version of a search. In this program, a list of three prompts will be assigned to a one-dimensional array C$(C), where C = 1, 2, or 3. The appropriate prompt will be displayed, based upon the choice of search selected by the user.

The program contains DATA representing names, hair color, eye color, and height in inches for ten hypothetical persons. The user of the program is given the choice of searching this DATA by hair color, eye color, or maximum height in inches. Following this option, a prompt is given for a key word upon which to base the search. A list of names, hair color, eye color, and height in inches matching the key word is displayed. The user is then given the option to do another search.

This program is similar in concept to PROGRAMs 4 and 5 in Chapter 3 in which DATA is READ and IF-THEN statements are used to check for a match with a key word INPUT by the user. However, the program is expanded in a number of ways. Note the comments alongside the program listing.

RUN from disk and refer to the listing and run of PROGRAM 6.

```
]LOAD PROGRAM 6
]LIST

10  REM  ===============
20  REM  PROGRAM 6 DESCRIPTION
30  REM  ===============
40  REM  "EXPANDED" SEARCH PROGRAM,
50  REM  ALLOWS SEARCH TO BE BASED UPON OPTIONAL
         KEYS.
60  REM  DATA CONTAINS NAME, HAIR COLOR, EYE
         COLOR, HEIGHT IN INCHES.
70  REM  LAST DATA ELEMENT SEQUENCE IS X,X,X,X.
80  REM  ===============
90  REM  VARIABLE DICTIONARY
100  REM ===============
110  REM C - SEARCH OPTION CHOICE
120  REM C$( ) - PROMPT FOR SEARCH KEY WORD
130  REM E$ - EYE COLOR
140  REM F - COUNTER FOR THE NUMBER OF "FINDS"
```

```
150  REM  H$ - HAIR COLOR
160  REM  I - LOOP COUNTER
170  REM  I$ - HEIGHT IN INCHES
180  REM  N$ - NAME
190  REM  S$ - SEARCH KEY WORD
200  REM  ===============
210  REM  ASSIGN THE OPTIONS
220  REM  ===============
230 C$(1) = "HAIR COLOR"
240 C$(2) = "EYE COLOR"
250 C$(3) = "MAXIMUM HEIGHT IN INCHES"
260  REM  ===============
270  REM  CLEAR THE SCREEN AND
280  REM  PRINT THE OPTIONS
290  REM  ===============
300 F = 0
310  HOME
320  PRINT "DO YOU WISH TO SEARCH BY:"
330  FOR I = 1 TO 3
340  PRINT "     ";I;" ";C$(I)
350  NEXT I
360  PRINT "ENTER THE NUMBER OF YOUR CHOICE";
370  INPUT C
380  REM  ===OUT OF RANGE CHECK===
390  IF C < 1 THEN 360
400  IF C > 3 THEN 360
410  PRINT C$(C);
420  INPUT S$
430  PRINT "*** SELECTION LIST ***"
440  PRINT "NAME","HAIR   EYES   HEIGHT"
450  PRINT "----","----   ----   ------"
460  FOR I = 1 TO 50
470  READ N$,H$,E$,I$
480  IF N$ = "X" THEN 640
490  REM  ===============
500  REM  GO TO THE APPROPRIATE
510  REM  LINE TO CHECK FOR MATCH
520  REM  OF OPTION SELECTED
530  REM  ===============
540  ON C GOTO 550,570,590
550  IF S$ = H$ THEN 600
560  GOTO 630
570  IF S$ = E$ THEN 600
580  GOTO 630
590  IF S$ < I$ THEN 630
600  PRINT N$,H$;"  ";E$;"  ";I$
610  REM   ===COUNT THE NUMBER FOUND===
620 F = F + 1
630  NEXT I
640  PRINT
```

230–250 assign values to string variables C$(1), C$(2), and C$(3).

330–350 print search choices.

370 assigns INPUT choice (1, 2, or 3) to variable C.

390–400 ensure that value INPUT is in range 1–3.

410 displays appropriate prompt, based upon value of C (e.g., if choice is 2, C has value 2; so C$(C) is C$(2), and "EYE COLOR" is printed at 410.)

420 INPUTs user's key word for search, assigning it to variable S$.

540 transfers execution to appropriate IF-THEN statement for match checking. [E.g., if C = 2 (from choice made at 370), transfer is to 570, where INPUT variable S$ is checked with E$, the eye color.]

In particular, note 590. BASIC allows comparison of string variables for "less than," "greater than," "not equal to," and so on. Just as a numeric value of 62 is less than a numeric value of 63, the character "A" is less than "B" which is less than "C", and so on.

```
650  PRINT "THIS SEARCH FOUND ";F;" ENTRIES."
660  PRINT "DO ANOTHER (Y OR N)";
670  INPUT Z$
680  IF Z$ <  > "Y" THEN 1000
690  RESTORE
700  GOTO 300
710  DATA  "MARY","BLACK","GREEN","62","BILL",
     "BROWN","BROWN","70"
720  DATA  "SUE","BLOND","GREEN","66","BOB",
     "BLACK","GREEN","72"
730  DATA  "JANE","BROWN","BROWN","69","JACK",
     "BLOND","GREEN","74"
740  DATA  "BETTY","BLACK","BLACK","66","FRED",
     "BLACK","BROWN","73"
750  DATA  "FRANCES","BLOND","BROWN","64","BUCK",
     "BLOND","GREEN","68"
760  REM  === ROOM FOR MORE DATA ===
999  DATA  "X","X","X","X"
1000  PRINT "*** SEARCH COMPLETED ***"
1010  END

]RUN
```

[Clear screen]

```
DO YOU WISH TO SEARCH BY:
     1 HAIR COLOR
     2 EYE COLOR
     3 MAXIMUM HEIGHT IN INCHES
ENTER THE NUMBER OF YOUR CHOICE?1
HAIR COLOR?BLOND
*** SELECTION LIST ***
NAME            HAIR   EYES   HEIGHT
----            ----   ----   ------
SUE             BLOND  GREEN  66
JACK            BLOND  GREEN  74
FRANCES         BLOND  BROWN  64
BUCK            BLOND  GREEN  68


THIS SEARCH FOUND 4 ENTRIES.
DO ANOTHER (Y OR N)?Y
```

[Clear screen]

```
DO YOU WISH TO SEARCH BY:
     1 HAIR COLOR
     2 EYE COLOR
     3 MAXIMUM HEIGHT IN INCHES
```

```
ENTER THE NUMBER OF YOUR CHOICE?5
ENTER THE NUMBER OF YOUR CHOICE?0
ENTER THE NUMBER OF YOUR CHOICE?2
EYE COLOR?HAZEL
*** SELECTION LIST ***
NAME            HAIR   EYES   HEIGHT
----            ----   ----   ------

THIS SEARCH FOUND 0 ENTRIES.
DO ANOTHER (Y OR N)?Y
```

[Clear screen]

```
DO YOU WISH TO SEARCH BY:
     1 HAIR COLOR
     2 EYE COLOR
     3 MAXIMUM HEIGHT IN INCHES
ENTER THE NUMBER OF YOUR CHOICE?2
EYE COLOR?BROWN
*** SELECTION LIST ***
NAME            HAIR   EYES   HEIGHT
----            ----   ----   ------
BILL            BROWN  BROWN  70
JANE            BROWN  BROWN  69
FRED            BLACK  BROWN  73
FRANCES         BLOND  BROWN  64

THIS SEARCH FOUND 4 ENTRIES.
DO ANOTHER (Y OR N)?Y
```

[Clear screen]

```
DO YOU WISH TO SEARCH BY:
     1 HAIR COLOR
     2 EYE COLOR
     3 MAXIMUM HEIGHT IN INCHES
ENTER THE NUMBER OF YOUR CHOICE?3
MAXIMUM HEIGHT IN INCHES?66
*** SELECTION LIST ***
NAME            HAIR   EYES   HEIGHT
----            ----   ----   ------
MARY            BLACK  GREEN  62
SUE             BLOND  GREEN  66
BETTY           BLACK  BLACK  66
FRANCES         BLOND  BROWN  64

THIS SEARCH FOUND 4 ENTRIES.
DO ANOTHER (Y OR N)?N
*** SEARCH COMPLETED ***
```

4.3.2 PROGRAM 7: Random Selection from a List

Another common use of one-dimensional arrays in instructional computing is in building lists of information for retrieval. These lists contain information, such as names, classification, and addresses; questions, answers, and hints for a given concept or topic; numerical data for analysis; and so on. Our next program illustrates one method of building lists and then randomly selecting items of information from them.

The primary purpose of this program is to demonstrate building one-dimensional string arrays (lists) and then randomly selecting from the lists. A user is given the option of making the length of the list from 3 to 15 elements. The lists, containing names and sex, are then made via INPUT statements. The user then has the option of selecting the number of names to be randomly selected from the list, with each randomly selected name appearing only once. This randomly selected list and the complete list of all names are printed.

RUN from disk and refer to the listing and run of PROGRAM 7.

```
]LOAD PROGRAM 7
]LIST

10  REM  ===============
20  REM   PROGRAM 7 DESCRIPTION
30  REM  ===============
40  REM  PROGRAM DEMONSTRATES THE
50  REM  USE OF 1-DIMENSIONAL STRING
60  REM  ARRAYS, DIM STATEMENTS,
70  REM  AND RANDOM SELECTION
80  REM  WITHOUT REPETITION.
90  REM  ===============
100  REM  VARIABLE DICTIONARY
110  REM  ===============
120  REM  N - NUMBER OF NAMES IN A LIST
     (VIA INPUT)
130  REM  N$( ) - HYPOTHETICAL NAMES
140  REM  S - NUMBER OF NAMES RANDOMLY
150  REM       SELECTED (VIA INPUT)
160  REM  S$( ) - SEX (MALE OR FEMALE)
170  REM  X - SOME RANDOM NUMBER
180  REM  Z(X) - FLAG FOR SELECTED
190  REM         RANDOM NUMBER
200  REM  IF Z(X) = 0, NUMBER HAS
210  REM  NOT BEEN RANDOMLY SELECTED
220  REM  IF Z(X) = 1, NUMBER HAS BEEN
230  REM  SELECTED AND PRINTED
240  REM  ===============
250  REM  DIMENSION THE VARIABLES.
260  REM  CLEAR THE SCREEN.
270  REM  GET SOME NAMES
```

10–290 briefly describe the purpose of the program and the variables used.

```
280  REM  AND SEX INPUT.
290  REM  ===============
300  DIM N$(15),S$(15),Z(15)
310  HOME
320  PRINT "HOW MANY NAMES DO YOU WISH TO ENTER"
330  PRINT "(AT LEAST 3 BUT NO MORE THAN 15)"
340  INPUT N
350  IF N < 3 OR N > 15 THEN 320
360  FOR I = 1 TO N
370  REM  ===INITIALIZE THE FLAG===
380 Z(I) = 0
390  PRINT "NAME NUMBER ";I;
400  INPUT N$(I)
410  PRINT "SEX (M OR F)";
420  INPUT S$(I)
430  NEXT I
440  PRINT "OF THE ";N;" NAMES, WHAT NUMBER"
450  PRINT "   DO YOU WISH TO RANDOMLY SELECT";
460  INPUT S
470  REM  ===ENOUGH AVAILABLE?===
480  IF S <  = N THEN 510
490  PRINT "IMPOSSIBLE!  YOU ONLY ENTERED ";N;"!"
500  GOTO 440
510  PRINT
520  PRINT S;" RANDOMLY SELECTED NAMES:"
530  PRINT "NUMBER","NAME","SEX"
540  PRINT "------","----","---"
550  REM  ===============
560  REM  SELECT THE NAMES
570  REM  AT RANDOM
580  REM  ===============
590  FOR I = 1 TO S
600  REM  ===GET A RANDOM X VALUE===
610 X =  INT (N * RND (1) + 1)
620  REM  ===HAS X APPEARED BEFORE?===
630  IF Z(X) = 0 THEN 660
640  PRINT "NUMBER ";X;" SELECTED AGAIN."
650  GOTO 610
660 Z(X) = 1
670  PRINT X,N$(X),S$(X)
680  NEXT I
690  PRINT
700  PRINT "THE COMPLETE LIST IS:"
710  PRINT "NUMBER","NAME","SEX"
720  PRINT "------","----","---"
730  FOR I = 1 TO N
740  PRINT I,N$(I),S$(I)
750  NEXT I
760  PRINT "* * * DONE * * *"
770  END
```

300 DIMensions (Section 4.4.1) length of one-dimensional arrays (lists) to maximum of 15 elements.

320–340 allow user to define number of elements in list.

350 shows how one IF-THEN statement may be used to check for both minimum *or* maximum values.

360–430 request a name [assigned to N$(I)] and sex [assigned to S$(I)] for each list element.

440–500 give user the option of defining number of random selections to be made from list just entered.

510–540 print heading for lists of randomly selected items.

590–680 print lists of randomly selected numbers, names, and sexes.

590 defines S (number of names selected) to be upper limit of loop.

610 assigns X some random value between 1 and N (number of names in complete list).

630 checks value of Z(X). If Z(X) = 0 [all values of Z(X) initialized to zero in 380], random number represented by X not previously used; so transfer is to 660, where Z(X) set to 1. If Z(X) = 1, random value X appeared previously; 640 then displays message illustrating that numbers may be randomly selected more than once. 650 then transfers execution to 610 where another random value is generated.

670 prints values of X, N$(X), S$(X). Loop continues at 680 until completion.

After randomly selected list is printed, 700–760 print complete list of names and sex; program then ends.

```
]RUN
```

[Clear screen]

```
HOW MANY NAMES DO YOU WISH TO ENTER
(AT LEAST 3 BUT NO MORE THAN 15)
?100
HOW MANY NAMES DO YOU WISH TO ENTER
(AT LEAST 3 BUT NO MORE THAN 15)
?7
NAME NUMBER 1?FRANK
SEX (M OR F)?M
NAME NUMBER 2?FRAN
SEX (M OR F)?F
NAME NUMBER 3?BILL
SEX (M OR F)?M
NAME NUMBER 4?BARB
SEX (M OR F)?F
NAME NUMBER 5?BUCK
SEX (M OR F)?M
NAME NUMBER 6?PAM
SEX (M OR F)?F
NAME NUMBER 7?GH
SEX (M OR F)?M
OF THE 7 NAMES, WHAT NUMBER
   DO YOU WISH TO RANDOMLY SELECT?10
IMPOSSIBLE!  YOU ONLY ENTERED 7!
OF THE 7 NAMES, WHAT NUMBER
   DO YOU WISH TO RANDOMLY SELECT?4

4 RANDOMLY SELECTED NAMES:
NUMBER          NAME            SEX
------          ----            ---
6               PAM             F
4               BARB            F
5               BUCK            M
7               GH              M

THE COMPLETE LIST IS:
NUMBER          NAME            SEX
------          ----            ---
1               FRANK           M
2               FRAN            F
3               BILL            M
4               BARB            F
5               BUCK            M
6               PAM             F
7               GH              M
* * * DONE * * *
```

4.3.3 Use of a Flag for Nonrepetitive Random Selection

In PROGRAM 7, all values for Z(X) are initialized to zero (statement 380) as the name and sex information is INPUT. A partial list of the information INPUT in the sample RUN of PROGRAM 7 would be:

X	N$(X)	S$(X)	Z(X)
1	Frank	M	0
2	Fran	F	0
3	Bill	M	0
4	Barb	F	0
.	.	.	.

If, for example, a random value for X is 3 (statement 610), Z(3) is set to 1, and the list would now be:

X	N$(X)	S$(X)	Z(X)
1	Frank	M	0
2	Fran	F	0
3	Bill	M	1
4	Barb	F	0
.	.	.	.

Since Z(3) is now equal to 1, any subsequent random value where X is equal to 3 would cause statements 630–650 to transfer execution back to statement 610, where another random value for X would be generated. Therefore, any randomly selected name in this program example will be displayed only one time.

4.4 BASIC STATEMENTS FOR THIS CHAPTER

4.4.1 Statement DIM

Purpose DIMensions (defines the size needed) for one-dimensional and two-dimensional arrays. On most BASIC systems, it is not necessary to use the DIM statement unless the array will contain more than ten elements. Most systems automatically allocate space for ten or less elements. However, it is good

programming practice to DIMension all arrays, even those containing ten or less elements.

Examples:
```
DIM N$(12)
```

(Dimensions space for a list of 12 string variables.)

```
DIM S(20,3)
```

(Dimensions space for a 20-row, 3-column table containing numeric data.)

4.4.2 Statement Pair GOSUB-RETURN

Purpose This statement pair is very useful for programs in which a sequence of statements is repeated several times throughout program execution. Whenever GOSUB is encountered, program execution is transferred to the statement number specified in the GOSUB. Execution continues from that statement until the RETURN statement is encountered. Execution is then transferred (RETURNed) to the statement number *immediately following* the GOSUB statement that caused the transfer in the first place.

A typical example in instructional computing would be an answer-checking sequence for student input in a program containing several questions. Rather than writing an identical answer-checking sequence for each question, it is written only once as a *subroutine*. GOSUB may then be used after each question to check the answer.

Example:
```
10 PRINT "CAPITAL OF TEXAS";
20 A$ = "AUSTIN"
30 GOSUB 200
40 PRINT "X MARKS THE --?--";
50 A$ = "SPOT"
60 GOSUB 200
70 GOTO 300
200 INPUT R$
210 IF R$ = A$ THEN 240
220 PRINT "NOPE...IT'S ";A$
230 GOTO 250
240 PRINT "OHHH, MARVELOUS!"
250 RETURN
300 PRINT,"BYE-BYE..."
310 END
```

Mentally execute this program before entering and RUNning it. Why is statement 70 needed?

4.4.3 PROGRAM 8: Question Sets and Hints, Using GOSUB-RETURN Statements

The following program illustrates the use of GOSUB-RETURN in a "muscle quiz." Note that alternate correct answers are used, note how the questions, additional hints, and answers are assigned and presented, and note that credit is

given for only those answers that are correct on the first try. Mentally outline the execution of this program carefully: Unlike previous programs, PROGRAM 8 does not have explanatory comments accompanying it.

RUN from disk and refer to the listing and run of PROGRAM 8.

```
]LOAD PROGRAM 8
]LIST

10  REM  ===============
20  REM   PROGRAM 8 DESCRIPTION
30  REM  ===============
40  REM  PROGRAM DEMONSTRATES THE USE OF
50  REM  GOSUB-RETURN, CHECKING FOR
60  REM  SYNONYMOUS CORRECT ANSWERS, AND
70  REM  THE USE OF ON-GOTO FOR HINTS.
80  REM  ===============
90  REM  VARIABLE DICTIONARY
100  REM  ===============
110  REM  C - CORRECT ANSWER COUNTER
120  REM  C$ - SYNON. CORRECT ANSWER
130  REM  D$ - SYNON. CORRECT ANSWER
140  REM  E$ - SYNON. CORRECT ANSWER
150  REM  F - FLAG FOR MISS ON 1ST TRY
160  REM  X - QUESTION NUMBER COUNTER
170  REM  ===============
180  HOME
190  PRINT "   M U S C L E   Q U I Z"
200  PRINT
210  REM  ===============
220  REM  PRINT THE QUESTION, DEFINE
230  REM  THE QUESTION NUMBER AND THE
240  REM  ANTICIPATED CORRECT ANSWERS. THEN
250  REM  GO TO THE SUBROUTINE FOR INPUT.
260  REM  ===============
270  REM
280  PRINT "WHAT IS THE LARGEST MUSCLE"
290  PRINT "   IN THE HUMAN BODY ";
300 X = 1
310 C$ = "GLUTEUS MAXIMUS"
320 D$ = "BUTTOCKS"
330 E$ = "DERRIERE"
340  GOSUB 5000
350  REM  ===============
360  REM  REPEAT THE QUESTION AND
370  REM  ANSWER SEQUENCE AGAIN
380  REM  ===============
```

```
390  PRINT "WHAT MUSCLE IS CONSIDERED BY SOME"
400  PRINT "   TO HAVE AN ORIGIN"
410  PRINT "BUT NO INSERTION ";
420 X = 2
430  REM  ===ONLY ONE ANSWER===
440 C$ = "TONGUE"
450 D$ = C$
460 E$ = C$
470  GOSUB 5000
480  REM  ===============
490  REM  REPEAT THE PROCESS
500  REM  FOR A FINAL QUESTION
510  REM  ===============
520  PRINT "WHAT MUSCLE HAS MADE MARK EDEN RICH,"
530  PRINT "   OLD MEN LEER"
540  PRINT "AND WEIGHT LIFTERS STRUT ";
550 X = 3
560 C$ = "PECTORAL"
570 D$ = "PECTORALIS MAJORA"
580 E$ = "PECTORALS"
590  GOSUB 5000
600  REM  ===============
610  REM  THERE'S PLENTY OF ROOM
620  REM  TO ADD MORE QUESTIONS AND
630  REM  ANSWERS FOLLOWING THE SAME
640  REM  SEQUENCE AS ABOVE.
650  REM  HINTS WOULD NEED TO BE
660  REM  ADDED TO THE SUBROUTINE.
670  REM  ===============
680  REM
690  REM  ===============
700  REM  STATEMENT 4990 IS NEEDED
710  REM  TO SKIP THE SUBROUTINE,
720  REM  GIVE THE PERFORMANCE SCORE,
730  REM  AND END THE PROGRAM.
740  REM  ===============
4990  GOTO 5270
5000  INPUT A$
5010  IF A$ = C$ THEN 5200
5020  IF A$ = D$ THEN 5200
5030  IF A$ = E$ THEN 5200
5040  REM  ===MISSED BEFORE? (F=1)===
5050  IF F = 1 THEN 5170
5060 F = 1
5070  REM  ===GIVE HINT FOR QUEST 1,2, OR 3===
5080  ON X GOTO 5090,5120,5140
5090  PRINT "SOME AUTHORITIES SAY WOMEN FIRST
      NOTICE"
5100  PRINT "THIS...  NOW TRY AGAIN..."
```

```
5110  GOTO 5000
5120  PRINT "ON SOME, IT WAGS ALOT.  NOW TRY..."
5130  GOTO 5000
5140  PRINT "PALM-TO-PALM PRESSURE DEVELOPS THIS"
5150  PRINT "MUSCLE.  TRY IT AGAIN..."
5160  GOTO 5000
5170  PRINT "A CORRECT ANSWER IS ";C$
5180  GOTO 5240
5190  REM  ===NO CREDIT GIVEN IF MISSED 1ST TRY===
5200  IF F = 1 THEN 5220
5210 C = C + 1
5220  PRINT "O.K."
5230  REM  ===SET F TO ZERO BEFORE NEXT QUESTION===
5240 F = 0
5250  PRINT
5260  RETURN
5270  PRINT
5280  PRINT "YOU GOT ";C;" CORRECT ON THE FIRST TRY."
5290  PRINT ,"THAT'S ALL..."
5300  END

]RUN
```

[Clear screen]

```
   M U S C L E   Q U I Z

WHAT IS THE LARGEST MUSCLE
   IN THE HUMAN BODY ?HEAD
SOME AUTHORITIES SAY WOMEN FIRST NOTICE
THIS...  NOW TRY AGAIN...
?BICEPS
A CORRECT ANSWER IS GLUTEUS MAXIMUS

WHAT MUSCLE IS CONSIDERED BY SOME
   TO HAVE AN ORIGIN
BUT NO INSERTION ?BICEPS
ON SOME, IT WAGS ALOT.  NOW TRY...
?TAIL
A CORRECT ANSWER IS TONGUE

WHAT MUSCLE HAS MADE MARK EDEN RICH,
   OLD MEN LEER
AND WEIGHT LIFTERS STRUT ?CHEST
PALM-TO-PALM PRESSURE DEVELOPS THIS
MUSCLE.  TRY IT AGAIN...
?PECTORALS
O.K.

YOU GOT 0 CORRECT ON THE FIRST TRY.
                THAT'S ALL...
```

```
]RUN
```

[Clear screen]

```
   M U S C L E  Q U I Z

WHAT IS THE LARGEST MUSCLE
   IN THE HUMAN BODY ?GLUTEUS BIGGEST
SOME AUTHORITIES SAY WOMEN FIRST NOTICE
THIS...  NOW TRY AGAIN...
?GLUTEUS MAXIMUS
O.K.

WHAT MUSCLE IS CONSIDERED BY SOME
   TO HAVE AN ORIGIN
BUT NO INSERTION ?TONGUE
O.K.

WHAT MUSCLE HAS MADE MARK EDEN RICH,
   OLD MEN LEER
AND WEIGHT LIFTERS STRUT ?PECTORALIS MAJORA
O.K.

YOU GOT 2 CORRECT ON THE FIRST TRY.
                THAT'S ALL...
```

4.5 POSERS AND PROBLEMS

1. Assume you have a class of 20 students and the semester test average and final exam scores for each. Outline the BASIC statements that would make a series of lists of this information.

2. Outline the BASIC statements that will read 3 scores for each of 25 students into a two-dimensional array.

3. Write a brief paragraph outlining the execution of the program consisting of the combined statements of Section 4.2.2.

4. Describe what would result from execution of the following BASIC statements:

```
10 DATA "TEXAS","OKLAHOMA","KANSAS","NEVADA","UTAH"
20 FOR I = 1 TO 5
30 READ S$(I)
40 NEXT I
50 FOR K = 1 TO 3
60 X = INT(5*RND(1)+1)
70 PRINT S$(X)
80 NEXT K
90 END
```

5. How would adding the following statements affect the execution of the program in Problem 4?

```
34 Z(I) = 0
64 IF Z(X) = 1 THEN 60
66 Z(X) = 1
```

6. Describe the execution of the program in Problem 5 if statement 50 is changed to

```
50 FOR K = 1 TO 6
```

Think this through carefully before entering and RUNning. Remember, CTRL-C (CONTROL and C keys depressed simultaneously) will halt a runaway program!

7. How would you modify the program in Section 4.4.2 to print a hint as a response for the first miss and give the correct answer on the second miss? [*Hint:* One way to do this is shown in PROGRAM 8. Another way is to assign the hint to, say, H$ and use an IF-THEN (IF F = 1 THEN...) to either give the hint (F = 0) or the correct answer (F = 1).]

8. Write a program that creates five random sentences from lists of subjects, verbs, and objects.

9. Write a program to choose and print four random numbers between 1 and 10 without repeating any number that has been printed.

"'Tis an old tale, and often told."
—Walter Scott

"And look before you, ere you leap;
For as you sow, y' are like to reap."
—Samuel Butler

Think About This (for Fun)

Four men are on a raft in the middle of the ocean. Each has one carton of cigarettes but no means whatsoever of lighting them. The smartest of the four, however, devotes his full mental prowess to the problem and, within minutes, all are smoking a cigarette. How was this accomplished?

Think About This (Seriously)

Could the role of the teacher change as computers become common in our schools? If so, how do you see this new role?

Chapter 5

Relax and Catch Your BASIC Breath

5.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

> Design, enter and RUN a minimum of three short programs using the BASIC statements summarized below.

5.2 BASIC STATEMENTS: A SUMMARY AND SOME TYPICAL USES

As an examination of any BASIC text or manual will show, there is more to the language than has been discussed so far. To this point, there has been presented

nothing more than just a brief introduction to the basics of BASIC. Although there is much more (some of which will be presented in later chapters), the statements and commands discussed to this point do provide a foundation for instructional computing applications.

This chapter summarizes these statements and provides a very general outline for their use in designing application programs. This summary must be general in illustrating applications since, as in any writing endeavor (programming or otherwise), the author's creativity is the limiting factor. The BASIC statements provide only the means by which programs can be constructed. The content, design strategies, effectiveness, and applicability of programs are the end result of creativity. Programs can only be as good—or as bad—as this factor.

Note: Many of the following examples are actually program fragments that may be referenced for future program development.

5.2.1 PRINT

Displays (outputs) information

Some typical uses	Example
Output text	`PRINT "WHAT'S YOUR NAME";`
Text + numeric variable	`PRINT "SCORE IS ";S;" PERCENT"`
Text + string variable	`PRINT "HELLO, ";N$;"!"`
Numeric with spacing	`PRINT A,B,C`
String (close-packed)	`PRINT N$;S$;A$`

5.2.2 LET

Assignment of a value to a variable

Some typical uses	Example
Counters	`C = C + 1`
Assigning correct answers	`A$ = "AUSTIN"` `A = Y * Z`
Assigning hints	`H$="'RIVER CITY'"`
Computation	`S = C * 100/Q`

5.2.3 INPUT

Receives information (input) from keyboard and assigns to defined variables

Some typical uses	Example
Numeric input	`INPUT N`
String input	`INPUT N$`
Combinations	`INPUT N$,N` `INPUT X,Y,Z`

5.2.4 GOTO

Unconditional branch (transfer of execution) to a specified statement number

Some typical uses	Example

Skipping statements	`100 GOTO 130`
	`110 PRINT "VERY GOOD!"`
	`120 C = C + 1`
	`130 PRINT "NEXT QUESTION..."`

Returning for input	`100 INPUT R`
	`⋮`
	`150 PRINT "NO, TRY AGAIN..."`
	`160 GOTO 100`

(*Note:* The ⋮ denotes omitted program segments.)

5.2.5 IF-THEN

Conditional branch to specified statement number if the defined condition is true

Some typical uses	Example

Answer checking	`100 IF R$=A$ THEN 130`
	`110 PRINT "NO, THE ANSWER IS ";A$`
	`120 GOTO 140`
	`130 PRINT "GOOD SHOW!"`

continued

Determining the sequence of execution

```
140 IF F = 1 THEN 160
150 C = C + 1
160 PRINT " NEXT QUESTION..."
 :
```

```
100 IF R < 40 THEN 140
110 IF R > 40 THEN 160
 :
140 PRINT "TOO LOW..."
 :
160 PRINT "TOO HIGH..."
```

```
100 IF F = 1 THEN 200
110 F = 1
120 PRINT "HERE'S A HINT..."
 :
200 PRINT "THE CORRECT ANSWER IS ";A
```

```
100 IF A$ = "STOP" THEN 800
 :
800 PRINT "HERE'S YOUR SCORE..."
```

5.2.6 ON-GOTO

Branch to a specified statement, based on the value of a defined variable or expression

Some typical uses	Example

Branch to a randomly selected question

```
 :
100 X = INT (5 * RND (1) + 1)
 :
150 ON X GOTO 200,300,400,500,600
 :
200 PRINT "QUESTION 1..."
 :
300 PRINT "QUESTION 2..."
 :
```

Branch to a hint for a given question

```
  .
  .
  .
100 PRINT "QUESTION 1..."
110 Q = 1
  .
  .
  .
500 INPUT R$
  .
  .
  .
550 PRINT "HERE'S A HINT..."
560 ON Q GOTO 700,800,900
  .
  .
  .
700 PRINT "HINT FOR QUESTION 1..."
710 GOTO 500
  .
  .
  .
```

5.2.7 DATA-READ (Statement pair)

Stores and assigns (READs) information to a defined variable

Some typical uses	**Example**

Assignment of question answer and hint	`100 REM *** Q$=QUES A$=ANS H$=HINT`
	`110 DATA "TEXAS","AUSTIN","RIVER CITY"`
	`  .`
	`  .`
	`  .`
	`200 READ Q$,A$,H$`
	`  .`
	`  .`
	`  .`

Assignment of numerical information	`100 DATA 90,76,55,70,88,93`
	`  .`
	`  .`
	`  .`
	`150 READ S1,S2,S3`
	`160 T = T + S1 + S2 + S3`
	`  .`
	`  .`
	`  .`
	`200 READ S1,S2,S3`
	`  .`
	`  .`
	`  .`

5.2.8 FOR-NEXT (Statement pair)

Repeats (loops) statement sequence between the FOR and the NEXT a defined number of times

Some typical uses

Assigning data to arrays

```
200 PRINT "NUMBER OF SCORES";
210 INPUT N
220 FOR I = 1 TO N
230 PRINT "SCORE";
240 INPUT S(I)
250 T = T + S(I)
260 NEXT I
270 A = T/N
280 PRINT "AVERAGE SCORE IS ";A
 :
```

```
100 DATA "QUES 1","ANS 1","HINT 1"
110 DATA "QUES 2","ANS 2","HINT 2"
 :
200 FOR I = 1 TO 10
210 READ Q$(I) ,A$(I) ,H$(I)
220 NEXT I
 :
```

Checking response to match any defined answer

```
 :
200 FOR I = 1 TO 10
210 IF R$ = A$(I) THEN 240
220 NEXT I
230 GOTO 500
240 PRINT "MATCHES ANS NO. ";I
 :
```

Defining the number of questions to be asked

```
 :
100 PRINT "HOW MANY DO YOU WANT";
110 INPUT P
120 IF P < 26 THEN 150
130 PRINT "THAT'S TOO MANY!"
140 GOTO 100
150 FOR I = 1 TO P
160 PRINT "PROBLEM NUMBER ";P
 :
300 NEXT I
 :
```

5.2.9 GOSUB-RETURN (Statement pair)

GOes to the statement number defining the SUBroutine, executes the statements in sequence until RETURN is executed, and then returns to the statements following the GOSUB

Some typical uses

Answer-checking sequence

```
 :
100 PRINT "QUESTION 1..."
105 A$ = "ANSWER 1"
110 GOSUB 2000
120 PRINT "QUESTION 2..."
125 A$ = "ANSWER 2"
130 GOSUB 2000
 :
2000 INPUT R$
2010 IF R$ = A$ THEN 2080
 :
2080 C = C + 1
2090 PRINT "HOT-DOGGIES!"
2100 RETURN
```

Generating random numbers

```
 :
100 GOSUB 500
110 PRINT "IF MASS = ";X;" AND"
120 PRINT "VOLUME = ";Y;", DENSITY =";
130 INPUT R
140 IF R = Z THEN 200
 :
500 X = INT (1000 * RND(1) + 1)/100
510 Y = INT (500 * RND(1) + 1)/10
520 Z = X/Y
530 RETURN
```

Generating random responses

```
10 DATA "GOOD","GREAT","WOW"
20 FOR I = 1 TO 3
30 READ R$(I)
40 NEXT I
 :
 :
500 GOSUB 750
 :
 :
750 X = INT(3 * RND(1) + 1)
760 PRINT R$(X);"!"
770 RETURN
```

5.3 A SUMMARY OF THE PURPOSES OF BASIC STATEMENTS

Although there is much more to the BASIC language, the statements summarized in this chapter are, nonetheless, the fundamental statements used in constructing instructional computing programs. In essence, these statements form the foundation upon which a program author has:

1. Some means of assigning and/or displaying values.
2. Some means of controlling the sequence of execution.
3. Some means of easing repetitious tasks.

Thus, most of the BASIC statements discussed to this point may be further summarized into three categories:

Assignment	**Control**	**Repetition**
PRINT	GOTO	FOR-NEXT
LET	IF-THEN	GOSUB-RETURN
INPUT	ON-GOTO	
DATA-READ		

This further generalization can be helpful in the initial design stages of program development. Once the category for a particular design task is identified, it becomes a matter of selecting the appropriate statements and defining their sequence of execution.

5.4 POSERS AND PROBLEMS

1. Assume a program is to be designed that gives credit for a correct answer only if it is answered correctly on the first attempt. Outline the statements needed to accomplish this.

2. Assume a program is to be designed that gives a hint on the first miss and the correct answer on the second miss. Outline the statements needed to accomplish this.

3. Assume 5 questions are to be randomly selected from a one-dimensional array containing 10 questions without repetition of any question during a given RUN. Outline the statements needed to accomplish this. (*Hint:* See Problems 4 and 5 in Chapter 4.)

4. Assume a program is to be designed that:

    1. Stores 15 questions, 15 answers, 15 hints, 3 positive responses, and 3 negative responses in one-dimensional arrays.

    2. Will ask a total of 8 questions.

    3. Will randomly select each question.

    4. Will not repeat any question.

    5. Will give a random positive response if correct.

    6. Will give a random negative response and an appropriate hint for the first miss.

    7. Will give the correct answer on the second miss.

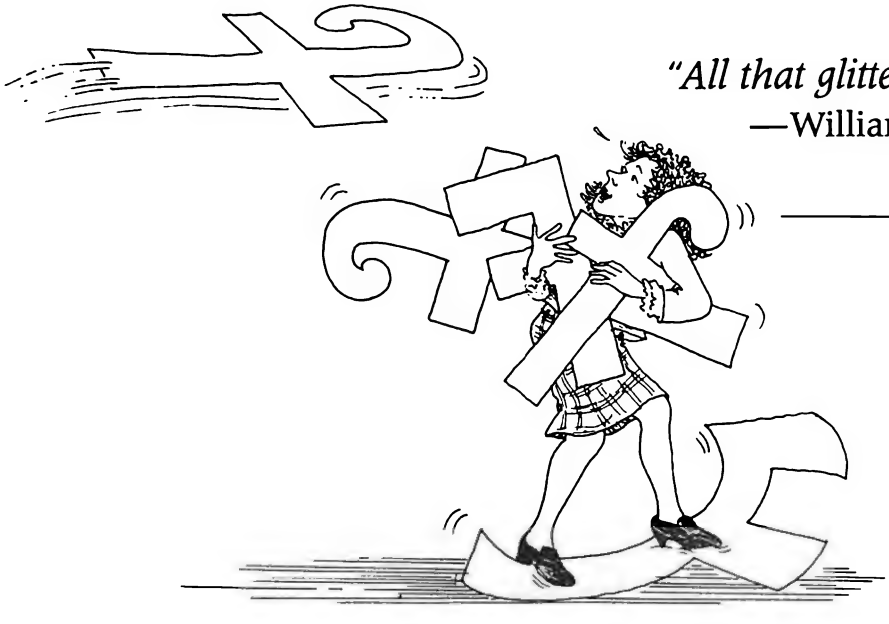
    8. Will give the number of correct answers at the conclusion of the interaction.

    Outline the statements needed to accomplish each of the above steps.

"The opportunities of man are limited only by his imagination. But so few have imagination that there are ten thousand fiddlers to one composer."
—Charles F. Kettering

"The most beaten paths are certainly the surest; but do not hope to scare up much game on them."
—André Gide

"All that glitters is not gold."
—William Shakespeare



Think About This (for Fun)

Read this sentence slowly: "Finished files are the result of years of scientific study combined with the experience of years." Now, once and only once, count out loud the F's (and f's) in that sentence. How many are there?

Think About This (Seriously)

Should at least one course in "Computer Literacy" be required for teacher certification in any area?

Chapter 6

Show and Tell

6.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Describe the purpose or application of instructional computing programs that are:
   a. problem solvers
   b. drill and practice
   c. tutorial (dialog)
   d. simulation
   e. testing
   (Sections 6.3–6.7).
2. Describe in outline form the sequence of execution for each of the example programs or program fragments in this chapter.

6.2 SOME EXAMPLE PROGRAMS AND PROGRAMMING STRATEGIES

One of the important factors determining the success or failure of a given human endeavor is the amount of imagination (originality, creativity, innovation, etc.) that goes into it. This applies not only to education in general and the instructional process in particular but also to the use of computers in instruction (in general) and the successful design and development of instructional computing programs (in particular).

In Chapter 8, specific steps will be discussed in which imagination will have an opportunity to spring forth. Before these steps are discussed, however, examine a few sample programs and program fragments that give an introduction to strategies and techniques for five methods in which instructional computing may be applied.

As these programs are examined, please keep in mind the quotations at the beginning of this chapter. These example programs *are* limited—by imagination and space. They are not meant to be the "well-beaten path" for those who choose to follow one. Also, they are certainly not meant to reflect the "gold" of instructional computing applications. However, they might plant an "imaginative seed" to allow one to reach heights of greater glory and reward in developing instructional computing programs.

Many of the examples are trivial in content. This is done intentionally because the content is not the point to be made: Rather, the programs illustrate some of the *strategies* that may be used in designing instructional computing programs. The content is left to the individual author who might use or expand upon these strategies.

Although some of the examples may be related to a specific discipline, this should be considered only as a "illustrative vehicle." In many cases, the pedagogical strategy used in the program could be applied in general, even though the content may be specific.

6.2.1 A Note About REM Statements

REM statements are very important for documenting a program listing. If carefully used in the program, they allow the reader to follow the sequence of program "events" with greater ease; their use makes the program more readable to the eye. For this reason, REM statements are extensively used in the following example programs. Hopefully, they will allow the user (particularly the beginning user) to better visualize a program's design and strategy.

This extensive use of REM statements may give the impression that the example programs are overly long and complicated. This is not the case. If the REM statements were removed, most of the example programs would be less than 50 to 75 statements long. Remember, then, that the REM statements are there to help explain the program as the listing is examined.

It should also be mentioned that REM statements, just like any other statements, require space in the system's memory. If the memory available in a system

is limited and the program design is lengthy, judicious use of REMs should be made.

Finally, and perhaps foremost, keep in mind that many of these example programs may be easily modified and expanded for actual class use. The REM statements in the listings will help explain how to do this.

6.3 PROBLEM-SOLVING APPLICATIONS

The heaviest use of instructional computing to date is that of problem solving—writing computer programs to solve specific discipline-oriented problems. This particular application, for all practical purposes, has no limits. It could be finding the roots of a quadratic equation in mathematics, calculating lunar orbits in physics, solving gas-law problems in chemistry, analyzing voting behavior in sociology, determining circulation trends in library science, and so on.

6.3.1 PROGRAM 9: Compound Interest

Most, if not all, problem-solving programs are based upon some formula or mathematical expression. Known parameters (elements) of the expression are input or read from data and the solution to an unknown parameter is calculated and output. PROGRAM 9 illustrates a business-oriented problem related to the return on invested capital. The known parameters are:

1. A given principal amount to be invested.
2. A given interest rate.
3. A given number of compounding periods per year.
4. A given number of years.

The future value is calculated and displayed, based on the following formula:

```
V = P * (1 + (I/N)) ^ (N*Y)
```

where

V = future value of the investment
P = principal amount invested
I = interest rate (decimal)
N = number of times the interest rate is compounded annually
Y = number of years the principal is invested

Remember, in the above formula and in statement 210 of the program, the caret is the Apple's way to "raise to the power of."

RUN from disk and refer to the listing and run of PROGRAM 9.

```
]LOAD PROGRAM 9
]LIST

10  REM   PROGRAM 9
20  REM  ===============
30  REM  PROBLEM SOLVING: THIS PROGRAM
40  REM  CALCULATES FUTURE VALUES OF
50  REM  INVESTMENTS. DEMOS 'FLASH'
60  REM  AND 'NORMAL' STATEMENTS.
70  REM  ===============
80  HOME
90  PRINT "   INVESTMENT FUTURE VALUES"
100  PRINT
110  PRINT "WHAT IS THE AMOUNT INVESTED";
120  INPUT P
130  PRINT "AT WHAT INTEREST RATE (%)";
140  INPUT I
150  PRINT "TIMES COMPOUNDED PER YEAR";
160  INPUT N
170 I = I / 100
180  PRINT "FOR HOW MANY YEARS";
190  INPUT Y
200  REM  ===FORMULA FOR CALC===
210 V = P * (1 + (I / N)) ^ (N * Y)
220  PRINT "ITS FUTURE VALUE WOULD BE ";
230  REM  ===SET THE FLASH DISPLAY===
240  FLASH
250  REM  ================
260  REM  MULTIPLY THE VALUE OF V BY 100; THEN
270  REM  GET THIS INTEGER VALUE; THEN DIVIDE
280  REM  BY 100.  THIS GIVES A VALUE TO 2 DECIMAL
     PLACES.
290  REM  ================
300  PRINT "$"; INT (V * 100) / 100
310  REM  ===SET BACK TO NORMAL DISPLAY===
320  NORMAL
330  PRINT
340  PRINT "CALCULATE ANOTHER (Y OR N)";
350  INPUT A$
360  IF A$ = "Y" THEN 100
370  END

]RUN PROGRAM 9
```

[Clear screen]

```
   INVESTMENT FUTURE VALUES

WHAT IS THE AMOUNT INVESTED?1000
AT WHAT INTEREST RATE (%)?16.5
```

```
TIMES COMPOUNDED PER YEAR?4
FOR HOW MANY YEARS?1
ITS FUTURE VALUE WOULD BE $1175.49

CALCULATE ANOTHER (Y OR N)?Y

WHAT IS THE AMOUNT INVESTED?1000
AT WHAT INTEREST RATE (%)?16.5
TIMES COMPOUNDED PER YEAR?4
FOR HOW MANY YEARS?20
ITS FUTURE VALUE WOULD BE $25374.71

CALCULATE ANOTHER (Y OR N)?Y

WHAT IS THE AMOUNT INVESTED?10000
AT WHAT INTEREST RATE (%)?12
TIMES COMPOUNDED PER YEAR?1
FOR HOW MANY YEARS?20
ITS FUTURE VALUE WOULD BE $96462.93

CALCULATE ANOTHER (Y OR N)

WHAT IS THE AMOUNT INVESTED?10000
AT WHAT INTEREST RATE (%)?12
TIMES COMPOUNDED PER YEAR?365
FOR HOW MANY YEARS?20
ITS FUTURE VALUE WOULD BE $110188.4

CALCULATE ANOTHER (Y OR N)?N
```

6.3.2 PROGRAM 10: Statistics

The formula for the *mean* of a set of scores is:

$$\text{Mean} = \frac{\text{Sum of scores}}{\text{Number of scores}}$$

The *variance* may be found from:

$$\text{Variance} = \frac{\text{Sum of squared differences between mean and scores}}{\text{Number of scores}}$$

The *standard deviation* of a set of scores is:

$$\text{Standard deviation} = \text{Square root of variance}$$

The *Z-score* may be found from:

$$Z = \frac{\text{Difference of score from mean}}{\text{Standard deviation}}$$

A program may be written to solve for these unknowns, given a set of scores.

RUN from disk and refer to the listing and run of PROGRAM 10.

```
]LOAD PROGRAM 10
]LIST

10  REM   PROGRAM 10
20  REM  ===============
30  REM  PROBLEM SOLVING: THIS PROGRAM
40  REM  CALCULATES MEAN, VARIANCE,
50  REM  STANDARD DEVIATION, AND
60  REM  Z-SCORES FOR A SET OF
70  REM  SCORES.
80  REM  ===============
90  REM
100  REM  ===============
110  REM  VARIABLE DICTIONARY
120  REM  ===============
130  REM  D - STANDARD DEVIATION
140  REM       (SQ. ROOT OF VARIANCE)
150  REM  D1( ) - DISTANCE OF GIVEN
160  REM            SCORE FROM MEAN
170  REM  M - MEAN OF SCORES
180  REM  S( ) - GIVEN SCORE
190  REM  T - CUMULATIVE TOTAL OF SCORES
200  REM  T1 - CUMULATIVE TOTAL OF THE SQUARE
210  REM        OF THE DISTANCE OF A GIVEN
220  REM        SCORE FROM THE MEAN
230  REM  V - VARIANCE
240  REM  ===============
250  REM
260  DIM S(100),D1(100)
270  HOME
280  PRINT "MEAN, VARIANCE, AND STANDARD"
290  PRINT "DEVIATION OF A SET OF SCORES"
300  PRINT
310  PRINT "ENTER THE SCORES. TO STOP,"
320  PRINT "  ENTER ANY NEGATIVE NO."
330  REM  ===ROOM FOR 100 SCORES===
340  FOR I = 1 TO 100
350  PRINT "SCORE (NEGATIVE TO STOP)";
360  INPUT S(I)
370  IF S(I) < 0 THEN 420
```

```
380  REM  ===CUMULATIVE SCORE TOTAL===
390 T = T + S(I)
400  NEXT I
410  GOTO 430
420 I = I - 1
430  PRINT
440  PRINT "MEAN","VAR","SD"
450  PRINT "----","---","--"
460  ===COMPUTE THE MEAN===
470 M = T / I
480  FOR J = 1 TO I
490  REM  ===GET DISTANCE OF SCORE FROM MEAN===
500 D1(J) = S(J) - M
510  REM  ===BUILD A CUMULATIVE TOTAL===
520 T1 = T1 + D1(J) ^ 2
530  NEXT J
540  REM  ===COMPUTE VARIANCE===
550 V = T1 / I
560  REM  ===COMPUTE STANDARD DEVIATION===
570 D = SQR (V)
580  REM  ===SET THE VALUES TO 2 DECIMAL PLACES===
590  PRINT  INT (M * 100) / 100, INT (V * 100) /
     100, INT (D * 100) / 100
600  PRINT
610  PRINT "     Z-SCORES"
620  PRINT "SCORE","Z-SCORE"
630  PRINT "-----","-------"
640  FOR J = 1 TO I
650  REM  ===============
660  REM  Z-SCORE IS THE DISTANCE OF A
670  REM  GIVEN SCORE FROM THE MEAN,
680  REM  DIVIDED BY THE STANDARD DEVIATION.
690  REM  ===============
700  PRINT S(J), INT (D1(J) / D * 100) / 100
710  NEXT J
720  PRINT
730  PRINT "ANALYZE ANOTHER SET (Y OR N)";
740  INPUT A$
750  IF A$ <  > "Y" THEN 790
760 T = 0
770 T1 = 0
780  GOTO 300
790  PRINT ,"BYE-BYE"
800  END

]RUN
```

[Clear screen]

```
MEAN, VARIANCE, AND STANDARD
DEVIATION OF A SET OF SCORES
```

```
ENTER THE SCORES. TO STOP,
  ENTER ANY NEGATIVE NO.
SCORE (NEGATIVE TO STOP)?100
SCORE (NEGATIVE TO STOP)?90
SCORE (NEGATIVE TO STOP)?80
SCORE (NEGATIVE TO STOP)?70
SCORE (NEGATIVE TO STOP)?60
SCORE (NEGATIVE TO STOP)?-1

MEAN            VAR             SD
----            ---             --
80              200             14.14

     Z-SCORES
SCORE           Z-SCORE
-----           -------
100             1.41
90               .7
80              0
70              _ .71
60              _1.42

ANALYZE ANOTHER SET (Y OR N)?Y

ENTER THE SCORES. TO STOP,
  ENTER ANY NEGATIVE NO.
SCORE (NEGATIVE TO STOP)?88
SCORE (NEGATIVE TO STOP)?91
SCORE (NEGATIVE TO STOP)?77
SCORE (NEGATIVE TO STOP)?55
SCORE (NEGATIVE TO STOP)?93
SCORE (NEGATIVE TO STOP)?85
SCORE (NEGATIVE TO STOP)?70
SCORE (NEGATIVE TO STOP)?-1

MEAN            VAR             SD
----            ---             --
79.85           158.97          12.6

     Z-SCORES
SCORE           Z-SCORE
-----           -------
88              .64
91              .88
77              -.23
55              -1.98
93              1.04
85              .4
70              -.79

ANALYZE ANOTHER SET (Y OR N)?N
                BYE-BYE
```

6.3.3 PROGRAM 11: File Maintenance

One problem that teachers face is ease of access to, and updating of, student records. PROGRAM 11 is one approach to letting a computer program do most of the work (the grades still have to be entered somehow—via a keyboard in this case). This program is included here to serve more as a utility program that the reader can use than as an illustrative example because some of the statements are more advanced than those of Chapters 1 through 5.

These new statements are needed to access another file, called TESTS, in which the record information is kept. The concept of this file is the same as that of DATA-READ program statements. However, the data (student records) are not accessed from DATA statements in the program but from the file TESTS. Use of such *text files* allows data to be updated for use in a program without having to rewrite or add DATA statements in the body of the program.

The contents of TESTS are numerical data in the sequence: n1,s1, s2,...,n2,s1,s2,...,etc. Here, n1 is the number of scores for the first student name and s1,s2,... are the scores for that student; n2 is the number of scores for the second student name and s1,s2,... are the scores for that student; and so on. An example might look like: 2,99,88,1,75,4,87,85,92,95,..., etc.

PROGRAM 11 appears to be lengthy but, again, this is due to extensive use of REM statements to help explain the program's execution. The program may be easily modified for actual class use by changing the DIM and DATA statements to meet the user's needs and running the program RECORD INITIALIZER from the diskette to erase the sample data in the file TESTS.

RUN from disk and refer to the listing and run of PROGRAM 11.

```
]LOAD PROGRAM 11
]LIST

10  REM   PROGRAM 11
20  REM  ==============
30  REM  PROBLEM SOLVING:
40  REM  THIS PROGRAM IS AN EXAMPLE OF RECORD
50  REM  KEEPING USING SEQUENTIAL "TEXT FILES."
60  REM  FIVE STUDENT NAMES ARE DATA ELEMENTS
70  REM  IN THE BODY OF THE PROGRAM.  THE
80  REM  NUMBER OF SCORES FOR EACH STUDENT AND
90  REM  THEIR RESPECTIVE SCORES ARE STORED
100  REM  SEQUENTIALLY IN THE TEXT FILE "TESTS"
110  REM  ON THE DISK.  THE PROGRAM MAY BE USED
120  REM  FOR REAL STUDENT RECORD KEEPING BY
130  REM  CHANGING THE DIM AND DATA STATEMENTS
140  REM  ACCORDINGLY, AND THEN RUNNING THE PROGRAM
150  REM  "RECORD INITIALIZER."
160  REM  ===============
170  REM  VARIABLE DICTIONARY
180  REM  ===============
```

```
190  REM  N$ - STUDENT NAME SOUGHT (VIA INPUT)
200  REM  N$( ) - STUDENT NAMES FROM PROGRAM DATA
210  REM           STATEMENTS
220  REM  N( ) - NUMBER OF SCORES FOR EACH
230  REM         STUDENT (FROM FILE "TESTS")
240  REM  P - NUMBER OF STUDENTS (FROM PROGRAM
250  REM      DATA STATEMENT)
260  REM  S( , ) - TWO-DIM ARRAY: ROW IS STUDENT
270  REM            NUMBER; COLUMN IS NUMBER OF
280  REM            SCORES FOR THAT STUDENT
290  REM  ===============
300  REM  PROGRAM EXAMPLE DIMENSIONS FOR A
310  REM  MAXIMUM OF 20 STUDENTS AND 8 SCORES.
320  REM  CHANGE DIM IF USED IN REAL CLASS.
330  REM  ===============
340  HOME : DIM S(20,8),N$(20),N(20)
350  PRINT "C L A S S  S C O R E  K E E P I N G"
360  REM  ===============
370  REM  DATA FOR NUMBER OF STUDENTS. CHANGE
380  REM  IF USED IN REAL CLASS.
390  REM  ===============
400  DATA  5
410  REM  ===============
420  REM  READ THE NUMBER OF STUDENTS; THEN
430  REM  STORE THE NAMES IN N$( ).
440  REM  ===============
450  READ P
460  REM  ===============
470  REM  5 STUDENT NAMES (CHANGE FOR REAL USE!)
480  DATA  "CANTOR","DARWIN","EDGAR","MCCARTHY","ZILLA"
490  REM  ===============
500  FOR I = 1 TO P
510  READ N$(I)
520  NEXT I
530  REM  ===============
540  REM  DEFINE D$ AS CONTROL-D (RULES OF THE
550  REM  GAME TO ACCESS TEXT FILE "TESTS")
560  REM  ================
570 D$ =  CHR$ (4)
580  REM  ===============
590  REM  THEN ISSUE "COMMANDS" TO OPEN THE
600  REM  FILE AND START READING THE DATA
610  REM  FROM IT. DATA ARE STORED IN THE
620  REM  SEQUENCE: (NUMBER OF SCORES FOR
630  REM  A STUDENT), (EACH SCORE FOR
640  REM  THAT STUDENT). N(I) IS THE NUMBER OF
650  REM  SCORES; S(I,J) IS THE SCORE.
660  REM  ===============
670  PRINT D$;"OPEN TESTS"
680  PRINT D$;"READ TESTS"
```

```
690  FOR I = 1 TO P
700  REM  ===GET THE NUMBER OF SCORES FROM FILE===
710  INPUT N(I)
720  REM  ===NOW GET EACH SCORE FROM FILE===
730  FOR J = 1 TO N(I)
740  INPUT S(I,J)
750  NEXT J
760  NEXT I
770  PRINT D$;"CLOSE TESTS"
780  REM  ================
790  REM  FINISHED READING DATA FROM FILE
800  REM  ================
810  PRINT
820  PRINT "DO YOU WANT TO:"
830  PRINT "1. ENTER NEW SCORES"
840  PRINT "2. RETRIEVE SCORES"
850  PRINT "3. STOP (ENTER 1-3)";
860  INPUT C
870  IF C = 1 THEN 910
880  IF C = 2 THEN 910
890  IF C = 3 THEN 1250
900  GOTO 810
910  PRINT
920  PRINT "STUDENT'S NAME (OR STOP)";
930  INPUT N$
940  IF N$ = "STOP" THEN 810
950  FOR I = 1 TO P
960  REM  ===MATCH FOUND WITH NAMES?===
970  IF N$ = N$(I) THEN 1010
980  NEXT I
990  PRINT N$;" IS NOT ON FILE!"
1000  GOTO 920
1010  IF C = 1 THEN 1190
1020  REM  ================
1030  REM  PRINT THE SCORES FOR THE STUDENT
1040  REM  ================
1050  PRINT "SCORES FOR ";N$(I);":"
1060  FOR J = 1 TO N(I)
1070  PRINT S(I,J);" ";
1080  REM  ===CUMULATIVE TOTAL FOR STUDENT===
1090 T = T + S(I,J)
1100  NEXT J
1110  REM  ===ILLEGAL TO DIVIDE BY ZERO===
1120  IF N(I) = 0 THEN 1140
1130  PRINT "  AVE = ";T / N(I)
1140 T = 0
1150  GOTO 910
1160  REM  ================
1170  REM  ADD MORE SCORES FOR STUDENTS
1180  REM  ================
```

```
1190  PRINT "NEXT SCORE FOR ";N$;
1200  REM  ===INCREASE THE SCORE COUNT BY 1===
1210 N(I) = N(I) + 1
1220  REM  ===STORE NEW SCORE IN ARRAY===
1230  INPUT S(I,N(I))
1240  GOTO 910
1250  PRINT
1260  REM === TURN ON PRINTER IF WANTED===
1262 PRINT "USE PRINTER (Y OR N)";
1264 INPUT Z$
1266 D$ = CHR$(4)
1268 IF Z$ = "Y" THEN PRINT D$; "PR#1"
1270  REM  ================
1280  REM  PRINT OUT CLASS RECORDS
1290  REM  ================
1300  PRINT "NAME"; TAB( 12);"SCORES"; TAB( 22);
      "AVERAGE"
1310  PRINT "----"; TAB( 12);"------"; TAB( 22);
      "-------"
1320  FOR I = 1 TO P
1330  FOR J = 1 TO N(I)
1340  REM  ===CUMULATIVE TOTAL FOR STUDENT===
1350 T = T + S(I,J)
1360  REM  ===CUMULATIVE TOTAL FOR CLASS===
1370 T1 = T1 + S(I,J)
1380  NEXT J
1390  REM  ===ILLEGAL TO DIVIDE BY ZERO===
1400  IF N(I) = 0 THEN 1430
1410  PRINT N$(I); TAB( 15);N(I); TAB
      ( 23);T / N(I)
1420  REM  ===CUMULATIVE TOTAL SCORE NUMBER===
1430 S1 = S1 + N(I)
1440 T = 0
1450  NEXT I
1460  PRINT
1470  PRINT "THE CLASS AVERAGE IS ";T1 / S1
1475  IF Z$ = "Y" THEN PRINT D$; "PR#0"
1480  REM  ================
1490  REM  WRITE ALL DATA WITH UPDATES
1500  REM  BACK ON THE FILE "TESTS."
1510  REM  ================
1520  PRINT D$;"OPEN TESTS"
1530  PRINT D$;"WRITE TESTS"
1540  FOR I = 1 TO P
1550  PRINT N(I)
1560  FOR J = 1 TO N(I)
1570  PRINT S(I,J)
1580  NEXT J
1590  NEXT I
1600  PRINT D$;"CLOSE TESTS"
```

```
1610  PRINT "      *** D O N E ***"
1620  END

]RUN
```

[Clear screen]

```
C L A S S  S C O R E  K E E P I N G

DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?5

DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?2

STUDENT'S NAME (OR STOP)?ZILLA
SCORES FOR ZILLA:
100 50 55    AVE = 68.3333334

STUDENT'S NAME (OR STOP)?BERGEN
BERGEN IS NOT ON FILE!
STUDENT'S NAME (OR STOP)?CANTOR
SCORES FOR CANTOR:
60 77 56 81    AVE = 68.5

STUDENT'S NAME (OR STOP)?TOPPER
TOPPER IS NOT ON FILE!
STUDENT'S NAME (OR STOP)?STOP

DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?1

STUDENT'S NAME (OR STOP)?CANTOR
NEXT SCORE FOR CANTOR?91

STUDENT'S NAME (OR STOP)?ZILLA
NEXT SCORE FOR ZILLA?72

STUDENT'S NAME (OR STOP)?STOP

DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?3
```

```
USE PRINTER (Y OR N)? N

NAME        SCORES     AVERAGE
----        ------     -------
CANTOR         5        73
DARWIN         8        79.25
EDGAR          4        59.5
MCCARTHY       2        67.5
ZILLA          4        69.25

THE CLASS AVERAGE IS 71.6956522
     *** D O N E ***
```

6.4 DRILL-AND-PRACTICE APPLICATIONS

Drill-and-practice programs are second only in use to problem-solving applications in instructional computing. This technique also has wide application in any area in which certain fundamental concepts require practice for mastery. This could be multiplication tables, chemical nomenclature, Latin-English word-root translations, state capitals, and so on.

Drill-and-practice programs are generally very straightforward: An introduction, usually including examples, is given; drill questions are presented (either linearly or by random selection); answers are entered and checked for accuracy; appropriate feedback is given; the next question is asked; and, at the end of the program, some form of performance report is given and, perhaps, recorded.

6.4.1 PROGRAM 12: Linear Selection of Drill Questions

Drill programs can be easily constructed with DATA-READ and FOR-NEXT statements. If the questions are to be linear, the DATA consists of question-answer pairs on any chosen topic that are READ as part of a FOR-NEXT question sequence. A skeleton program might be:

```
10 DATA["Question 1","Answer 1","Question 2", etc.]
 .
 .
 .
200 PRINT "[Introductory statements, examples, etc.]"
 .
 .
 .
500 FOR I = 1 TO [Number of questions to ask]
510 READ Q$,A$
520 PRINT Q$;
530 INPUT R$
540 IF R$ = A$ THEN 570
```

```
550 PRINT "A CORRECT ANSWER IS ";A$
560 GOTO 590
570 PRINT "EXCELLENT!"
580 C = C + 1
590 NEXT I
600 PRINT "YOU ANSWERED ";C;" QUESTIONS CORRECTLY!"
610 END
```

In the following program on state capitals (PROGRAM 12), note the use of the RND(1) function to "flip a coin" to determine if the state or the capital is to be asked as a question. In the normal READ sequence (as defined in this program), Q$ (the question) contains the state and A$ (the answer) contains the capital. If the question-answer values are to be reversed, the contents of the variables Q$ and A$ must be switched. This is accomplished in statements 440–460 by the use of a dummy variable, D$, to hold the original question (value of Q$) as the switching process is done.

Again, please remember that by just changing the contents of the DATA statements, it is possible to make the program more than just a trivial drill on state capitals!

RUN from disk and refer to the listing and run of PROGRAM 12.

```
]LOAD PROGRAM 12
]LIST

10  REM    PROGRAM 12
20  REM  ===============
30  REM   DRILL AND PRACTICE: THIS
40  REM  PROGRAM DEMOS THE LINEAR QUESTION-
50  REM  AND-ANSWER SEQUENCE VIA DATA-READ,
60  REM  PLUS SWITCHING A QUESTION FOR AN
70  REM  ANSWER AND AN ANSWER FOR A QUESTION.
80  REM
90  REM  ===============
100  REM  VARIABLE DICTIONARY
110  REM  ===============
120  REM  A$ - ANSWER (FROM DATA-READ)
130  REM  C - NUMBER CORRECT COUNTER
140  REM  D$ - HOLDS ORIGINAL QUESTION IN
150  REM       QUESTION/ANSWER SWITCHING
160  REM  Q$ - QUESTION (FROM DATA-READ)
170  REM  R$ - USER RESPONSE (VIA INPUT)
180  REM  ===============
190  DATA "TEXAS","AUSTIN","ARKANSAS","LITTLE ROCK"
200  DATA  "NEW MEXICO","SANTA FE","OKLAHOMA"
210  DATA  "OKLAHOMA CITY","OREGON","SALEM"
220  HOME
230  REM  ===INTRODUCTION===
```

```
240  PRINT "STATE CAPITAL DRILL"
250  PRINT
260  PRINT "IF I GIVE THE STATE, YOU GIVE"
270  PRINT "THE CAPITAL; IF I GIVE THE"
280  PRINT "CAPITAL, YOU GIVE THE STATE."
290  PRINT
300  FOR I = 1 TO 5
310  READ Q$,A$
320  PRINT
330  REM  ==='FLIP' A COIN===
340 X = INT (2 *  RND (1) + 1)
350  IF X = 2 THEN 490
360  REM  ===DO THE SWITCH IF X IS 1===
370  REM  ===============
380  REM  HERE'S THE SWITCH...STORE Q$ IN
390  REM  D$ TEMPORARILY, PUT THE ANSWER IN
400  REM  Q$ (NOW THE ANSWER IS THE QUESTION), AND
410  REM  THEN GET THE ANSWER FROM D$
420  REM  (NOW THE ORIGINAL QUESTION IS THE ANSWER).
430  REM  ===============
440 D$ = Q$
450 Q$ = A$
460 A$ = D$
470  REM  ===SWITCH COMPLETED===
480  REM  ===NOW ASK THE QUESTION===
490  PRINT Q$;
500  INPUT R$
510  IF R$ = A$ THEN 540
520  PRINT "A CORRECT ANSWER IS ";A$
530  GOTO 560
540  PRINT "GREAT!"
550 C = C + 1
560  NEXT I
570  PRINT "YOU GOT ";C;" CORRECT!"
580  END

]RUN PROGRAM 12
```

[Clear screen]

```
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

TEXAS?AUSTIN
GREAT!

ARKANSAS?LITTLE ROCK
GREAT!
```

```
NEW MEXICO?SACRAMENTO
A CORRECT ANSWER IS SANTA FE

OKLAHOMA CITY?OKLAHOMA
GREAT!

OREGON?PORTLAND
A CORRECT ANSWER IS SALEM
YOU GOT 3 CORRECT!

]RUN
```

[Clear screen]

```
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

TEXAS?AUSTIN
GREAT!

ARKANSAS?LITTLE ROCK
GREAT!

NEW MEXICO?SANTA FE
GREAT!

OKLAHOMA CITY?OKLAHOMA
GREAT!

SALEM?OREGON
GREAT!
YOU GOT 5 CORRECT!
```

6.4.2 PROGRAM 13: Random Selection of Drill Questions

If the questions are to be randomly selected from a bank of data elements, the DATA are READ into one-dimensional arrays prior to presentation of the question sequence:

```
10 DIM Q$([Number of questions in bank]),A$( ),Z( )
20 DATA ["Question ","Answer 1 ","Question 2", etc.]
.
.
.
160 FOR I = 1 TO [Number of questions in bank]
170 READ Q$(I),A$(I)
180 Z(I) = 0
```

```
190 NEXT I
200 PRINT "[Introductory statements, examples, etc.]"
⋮
500 FOR I = 1 TO [Number of questions to be asked]
505 REM RANDOMLY SELECT A QUESTION NUMBER
510 J = INT([Number of questions in bank]*RND(1) + 1)
515 REM HAS THIS NUMBER ALREADY BEEN SELECTED?
520 IF Z(J) = 1 THEN 510
530 Z(J) = 1
540 PRINT Q$(J);
550 INPUT R$
560 IF R$ = A$(J) THEN 590
570 PRINT "A CORRECT ANSWER IS ";A$(J)
580 GOTO 610
590 PRINT "GREAT!"
600 C = C + 1
610 NEXT I
620 PRINT "YOU ANSWERED";C;"QUESTIONS CORRECTLY!"
630 END
```

Note in the example, PROGRAM 13, that only 3 of the 5 possible questions are randomly selected. In general, it is good practice to have approximately 25% more questions in the bank than are to be asked by random selection. This will reduce the time required for the program to find a question that has not been asked previously.

RUN from disk and refer to the listing and run of PROGRAM 13.

```
]LOAD PROGRAM 13
]LIST

10  REM    PROGRAM 13
20  REM  ===============
30  REM   DRILL AND PRACTICE: THIS
40  REM  PROGRAM DEMOS RANDOM SELECTION OF
50  REM  QUESTIONS/ANSWERS FROM ONE-DIM ARRAYS
60  REM  WITHOUT REPEATING ANY QUESTION, PLUS
70  REM  SWITCHING THE QUESTION-ANSWER.
80  REM  ===============
90  REM  VARIABLE DICTIONARY
100  REM  ===============
110  REM  A$(J) - ANSWER TO QUESTION
120  REM  D$ - HOLDS QUESTION TEMPORARILY
130  REM        IN QUES/ANSWER SWITCHING
140  REM  J - RANDOM INTEGER VALUE
150  REM  Q$(J) - RANDOM QUESTION FROM LIST
160  REM  Z(J) - FLAG FOR SELECTED INTEGER
170  REM  ===============
180  DIM Q$(5),A$(5),Z(5)
```

```
190  DATA  "TEXAS","AUSTIN","ARKANSAS","LITTLE ROCK"
200  DATA  "NEW MEXICO","SANTA FE","OKLAHOMA"
210  DATA  "OKLAHOMA CITY","OREGON","SALEM"
220  HOME
230  REM  ===INTRODUCTION===
240  PRINT "STATE CAPITAL DRILL"
250  PRINT
260  PRINT "IF I GIVE THE STATE, YOU GIVE"
270  PRINT "THE CAPITAL; IF I GIVE THE"
280  PRINT "CAPITAL, YOU GIVE THE STATE."
290  PRINT
300  REM  ===STORE THE QUESTIONS/ANSWERS===
310  FOR I = 1 TO 5
320  READ Q$(I),A$(I)
330  NEXT I
340  REM  ===ASK ONLY 3 OF THE POSSIBLE 5===
350  FOR I = 1 TO 3
360  PRINT
370  REM  ===SELECT A QUESTION NUMBER===
380 J =  INT (5 *  RND (1) + 1)
390  REM  ===HAS IT BEEN SELECTED BEFORE?===
400  IF Z(J) = 1 THEN 380
410  REM  ===FLAG J AS A SELECTED NUMBER===
420 Z(J) = 1
430  REM
440  REM  ==='FLIP' A COIN===
450 X =  INT (2 *  RND (1) + 1)
460  IF X = 2 THEN 520
470  REM  ===DO THE SWITCH IF X IS 1===
480 D$ = Q$(J)
490 Q$(J) = A$(J)
500 A$(J) = D$
510  REM  ===ASK THE QUESTION===
520  PRINT Q$(J);
530  INPUT R$
540  IF R$ = A$(J) THEN 570
550  PRINT "A CORRECT ANSWER IS ";A$(J)
560  GOTO 590
570  PRINT "GREAT!"
580  C = C + 1
590  NEXT I
600  PRINT "YOU GOT ";C;" CORRECT!"
610  END
```

```
]RUN PROGRAM 13
```

[Clear screen]

```
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

SALEM?NEW MEXICO
A CORRECT ANSWER IS OREGON

AUSTIN?NEVADA
A CORRECT ANSWER IS TEXAS

NEW MEXICO? SANTA FE
GREAT!
YOU GOT 1 CORRECT!

]RUN
```

[Clear screen]

```
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

SANTA FE?NEW MEXICO
GREAT!

TEXAS?AUSTIN
GREAT!

OREGON?SALEM
GREAT!
YOU GOT 3 CORRECT!
```

6.4.3 PROGRAM 14: User Options and Random Positive Feedback

Use of GOSUB-RETURN routines makes development of linear drill-and-practice programs a simple task. The minimum needed could be:

- Present statements for the introduction, examples, and question content:

```
10 PRINT "[Introductory statements]"
.
.
.
100 PRINT "[Presenting examples]"
.
.
.
200 PRINT "[Ask question 1]'
```

- Assign a correct answer to a variable:

```
210 A$="[Answer to question 1]"
```

- Go to the answer-checking subroutine:

```
220 GOSUB 10000
```

- Present the next question following RETURN:

```
230 PRINT "[Ask question 2]"
```

- Assign answer to a variable:

```
240 A$="[Answer to question 2]"
```

- Go to the subroutine again:

```
250 GOSUB 10000
.
.
.
```

- The subroutine allows for answer input:

```
10000 INPUT R$
```

- Checks for accuracy:

```
10010 IF R$=A$ THEN 10040
```

- Presents a correct answer if missed:

```
10020 PRINT "A CORRECT ANSWER IS ";A$
```

- Returns to the next question:

```
10030 GOTO 10060
```

- Gives a positive feedback if correct:

```
10040 PRINT "VERY GOOD!"
```

- Increases a number-correct counter by 1:

```
10050 C = C + 1
```

- And asks the next question:

```
10060 RETURN
```

These statements outline a general design sequence. But, to a user, a drill-and-practice program that just asks a question, says "CORRECT" or "INCORRECT," and then asks the next question can be awfully boring. However, we can liven up the program by giving the user some options like: SKIP (a question), ANSWER (to a question), and STOP (at will). We can also randomize the positive feedback and use the INVERSE statement for emphasis in asking a question. Examples of these types of additions are shown in PROGRAM 14, which is a drill on the parts of a sentence.

Some comments about PROGRAM 14: Just when you thought you knew a bit about BASIC, along comes this program that's sooooo long! Don't tear your hair! Without the REM statements explaining the program's execution and the PRINT statements giving the introduction and examples, the program consists of only 73 statements. The key points brought out in these 73 statements are: the use of subroutines for repetitive processes; control and appearance of the screen display; and ease of expanding a program once these processes are defined.

Examination of the listing will show that the introduction and the sentences composing the questions (up to eight words per sentence) may be changed to a user's own choosing and that many more sentences may be added with ease. So relax. Carefully examine the method by which a sentence is defined (e.g., statements 710–1010) and the subroutines beginning at statements 4500 and 5000. These sections are the crux of comprehending this program.

RUN from disk and refer to the listing and run of PROGRAM 14.

```
]LOAD PROGRAM 14
]LIST

10  REM    PROGRAM 14
20  REM  ===============
30  REM  DRILL AND PRACTICE: THIS PROGRAM
40  REM  DEMOS RANDOM POSITIVE FEEDBACK
50  REM  FROM A ONE-DIM ARRAY, GIVING
60  REM  USER OPTIONS FOR PROGRAM CONTROL,
70  REM   AND THE USE OF THE INVERSE STATEMENT
80  REM  FOR EMPHASIS IN QUESTION SEQUENCE.
90  REM  TWO SUBROUTINES ARE USED.
100  REM  ===============
110  REM  VARIABLE DICTIONARY
120  REM  ===============
130  REM  A$ - ANTICIPATED CORRECT ANSWER
140  REM  C - NUMBER-CORRECT COUNTER
150  REM  F$( ) - RANDOM POSITIVE FEEDBACK
160  REM  R$ - USER INPUT
```

```
170  REM  W - NUMBER OF WORDS IN A GIVEN SENTENCE
180  REM  W$( ) - WORDS OF THE SENTENCE
190  REM  X - QUESTION (WORD) COUNTER
200  REM  Y - WORD NO. IN SENTENCE TO EMPHASIZE
210  REM  ===============
220  DIM F$(4),W$(8)
230  REM  ===POSITIVE FEEDBACK CHOICES===
240  DATA  "WELL DONE","MARVELOUS","THAT'S IT","VERY GOOD"
250  REM  ===STORE FEEDBACK CHOICES===
260  FOR I = 1 TO 4
270  READ F$(I)
280  NEXT I
290  REM  ===GIVE THE INTRODUCTION===
300  HOME
310  PRINT "     INTRODUCTION"
320  PRINT
330  PRINT "THIS IS A SHORT DRILL ON"
340  PRINT "SENTENCE STRUCTURE.  I'LL"
350  PRINT "PRESENT A COMPLETE SENTENCE"
360  PRINT "AND YOU ARE ASKED TO IDENTIFY"
370  PRINT "EACH PART OF THAT SENTENCE,"
380  PRINT "WORD BY WORD."
390  PRINT
400  REM  ===USER CONTROLS WHEN TO GO===
410  PRINT "READY FOR MORE";
420  INPUT Z$
430  HOME
440  PRINT "   *** EXAMPLE ***"
450  PRINT
460  PRINT "THE DOG BITES."
470  PRINT
480  PRINT "FIRST I'LL EMPHASIZE THE WORD 'THE'"
490  PRINT "AND YOU SHOULD IDENTIFY 'THE'"
500  PRINT "AS AN ARTICLE. NEXT I'LL"
510  PRINT "EMPHASIZE THE WORD 'DOG' WHICH SHOULD"
520  PRINT "BE IDENTIFIED AS THE SUBJECT."
530  PRINT "FINALLY, I'LL EMPHASIZE 'BITES'"
540  PRINT "WHICH SHOULD BE IDENTIFIED AS"
550  PRINT "A VERB."
560  PRINT
570  PRINT "SHALL I GO ON";
580  REM  ===USER CONTROLS WHEN TO GO===
590  INPUT Z$
600  HOME
610  PRINT
620  PRINT "ALSO, KNOW THAT YOU MAY SKIP"
630  PRINT "A QUESTION BY ENTERING 'SKIP',"
640  PRINT "RECEIVE A CORRECT ANSWER BY"
650  PRINT "ENTERING 'ANSWER', OR STOP AT"
660  PRINT "ANYTIME BY ENTERING 'STOP'."
```

```
670  PRINT
680  PRINT "ARE YOU READY FOR THE"
690  PRINT "FIRST SENTENCE";
700  INPUT Z$
710  REM  ===============
720  REM  THE DATA CONTAINS THE WORDS
730  REM  OF THE SENTENCE. 'W' IS THE
740  REM  NUMBER OF THE WORDS IN THE
750  REM  SENTENCE FOR THE READING LOOP.
760  REM  ===============
770  DATA  "JACK","LOVES","MARY."
780 W = 3
790  REM  ===============
800  REM  GO TO THE SUBROUTINE TO
810  REM  STORE THE WORDS IN W$( ).
820  REM  THE MAXIMUM NO. OF WORDS IS 8.
830  REM  ===============
840  GOSUB 4500
850  REM  ===============
860  REM  ASSIGN THE CORRECT ANSWER TO A$
870  REM  AND THE WORD NUMBER IN THE SENTENCE
880  REM  TO INVERSE FOR EMPHASIS TO Y, THEN
890  REM  GO TO THE QUES/ANSWER SUBROUTINE.
900  REM  ===============
910 Y = 1
920 A$ = "SUBJECT"
930  GOSUB 5000
940  REM  ===REPEAT THE PROCESS FOR THE NEXT WORD===
950 Y = 2
960 A$ = "VERB"
970  GOSUB 5000
980  REM  ===REPEAT THE PROCESS===
990 Y = 3
1000 A$ = "DIRECT OBJECT"
1010  GOSUB 5000
1020  REM  ===============
1030  REM  REPEAT THE PROCESS FOR
1040  REM  THE NEXT SENTENCE
1050  REM  ================
1060  DATA  "THE","CAT","SCRATCHED","MIKE."
1070 W = 4
1080  REM  ===GO TO THE STORING SUBROUTINE===
1090  GOSUB 4500
1100  REM  ===ASSIGN THE ANS. AND WORD NO. TO EMPHASIZE===
1110 Y = 1
1120 A$ = "ARTICLE"
1130  GOSUB 5000
1140 Y = 2
1150 A$ = "SUBJECT"
1160  GOSUB 5000
```

```
1170 Y = 3
1180 A$ = "VERB"
1190  GOSUB 5000
1200 Y = 4
1210 A$ = "DIRECT OBJECT"
1220  GOSUB 5000
1230  REM  ==============
1240  REM  ADDITIONAL SENTENCES MAY
1250  REM  BE ADDED BELOW, FOLLOWING
1260  REM  THE SAME SEQUENCE AS ABOVE.
1270  REM  ==============
4000  PRINT
4010  PRINT "THAT'S ALL FOR TODAY..."
4020  GOTO 5540
4100  REM  =============
4110  REM  THE 4500 SUBROUTINE READS THE WORDS
4120  REM  OF A GIVEN SENTENCE INTO AN
4130  REM  ARRAY SO THAT THE SENTENCE MAY
4140  REM  BE PRINTED LATER, WORD-BY-WORD,
4150  REM  AND THE APPROPRIATE WORD "INVERSED."
4160  REM  =============
4500  HOME
4510  FOR I = 1 TO W
4520  READ W$(I)
4530  NEXT I
4540  REM  =============
4550  REM  NOW THAT THE SENTENCE WORDS HAVE
4560  REM  BEEN STORED, RETURN AND START
4570  REM  THE QUESTION/ANSWER SEQUENCE.
4580  REM  =============
4590  RETURN
4600  REM  ==============
4610  REM  THE 5000 SUBROUTINE PRINTS THE
4620  REM  SENTENCE WORD-BY-WORD, INVERSING
4630  REM  THE WORD CORRESPONDING TO THE
4640  REM  QUESTION.  USER INPUT IS CHECKED
4650  REM  FIRST FOR AN 'OPTION' MATCH THEN
4660  REM  FOR THE CORRECT ANSWER. RANDOM
4670  REM  FEEDBACK IS GIVEN FOR CORRECT ANSWERS.
4680  REM  =============
5000  PRINT
5010  REM  ===START PRINTING THE WORDS===
5020  FOR I = 1 TO W
5030  REM  ===ADD 1 TO THE X COUNTER===
5040 X = X + 1
5050  REM  ===IS X EQUAL TO Y?  INVERSE IF SO===
5060  IF X = Y THEN 5090
5070  PRINT W$(I);" ";
5080  GOTO 5120
5090  INVERSE
```

```
5100  PRINT W$(I);" ";
5110  NORMAL
5120  NEXT I
5130  PRINT
5140  PRINT
5150  PRINT "YOUR ANSWER";
5160  INPUT R$
5170  REM  ===============
5180  REM  CHECK FIRST FOR 'OPTION' INPUT
5190  REM  ===============
5200  IF R$ = "SKIP" THEN 5490
5210  IF R$ = "ANSWER" THEN 5270
5220  IF R$ = "STOP" THEN 5540
5230  REM  ===============
5240  REM  THEN CHECK FOR A CORRECT ANSWER
5250  REM  ===============
5260  IF R$ = A$ THEN 5370
5270  PRINT "A CORRECT ANSWER IS ";A$
5280  REM  ================
5290  REM  HOLD THE CORRECT ANSWER UNTIL
5300  REM  THE USER IS READY TO CONTINUE.
5310  REM  ================
5320  PRINT
5330  PRINT "ARE YOU READY";
5340  INPUT Z$
5350  GOTO 5490
5360  REM  ===GET A RANDOM NUMBER (4-1)===
5370 F = INT (4 *  RND (1) + 1)
5380  REM  ===PRINT THE RANDOM FEEDBACK===
5390  PRINT F$(F);"!"
5400  REM  ===ADD 1 TO THE NUMBER CORRECT===
5410 C = C + 1
5420  REM  ===============
5430  REM  LET THE SYSTEM "COUNT" TO 2000 SO THAT
5440  REM  THE DISPLAY WILL REMAIN 2-3 SECONDS
5450  REM  ===============
5460  FOR K = 1 TO 2000
5470  NEXT K
5480  REM  ===SET X TO ZERO BEFORE NEXT
      QUESTION===
5490 X = 0
5500  RETURN
5510  REM  =============
5520  REM  GIVE SCORE AND END PROGRAM
5530  REM  =============
5540  PRINT
5550  PRINT "YOU GOT ";C;" CORRECT ANSWER(S)."
5560  END
```

```
]RUN
```

[Clear screen]

```
     INTRODUCTION

THIS IS A SHORT DRILL ON
SENTENCE STRUCTURE.  I'LL
PRESENT A COMPLETE SENTENCE
AND YOU ARE ASKED TO IDENTIFY
EACH PART OF THAT SENTENCE,
WORD BY WORD.

READY FOR MORE?OK
```

[Clear screen]

```
   *** EXAMPLE ***

THE DOG BITES.

FIRST I'LL EMPHASIZE THE WORD 'THE'
AND YOU SHOULD IDENTIFY 'THE'
AS AN ARTICLE.  NEXT I'LL
EMPHASIZE THE WORD 'DOG' WHICH SHOULD
BE IDENTIFIED AS THE SUBJECT.
FINALLY, I'LL EMPHASIZE 'BITES'
WHICH SHOULD BE IDENTIFIED AS
A VERB.

SHALL I GO ON?YES ...GO ON
```

[Clear screen]

```
ALSO, KNOW THAT YOU MAY SKIP
A QUESTION BY ENTERING 'SKIP',
RECEIVE A CORRECT ANSWER BY
ENTERING 'ANSWER', OR STOP AT
ANYTIME BY ENTERING 'STOP'.

ARE YOU READY FOR THE
FIRST SENTENCE?YES
```

[Clear screen]

```
JACK LOVES MARY.

YOUR ANSWER?ANSWER
A CORRECT ANSWER IS SUBJECT

ARE YOU READY?OK
```

```
JACK LOVES MARY.

YOUR ANSWER?SKIP

JACK LOVES MARY.

YOUR ANSWER?SUBJECT
A CORRECT ANSWER IS DIRECT OBJECT

ARE YOU READY?OK
```

[Clear screen]

```
THE CAT SCRATCHED MIKE.

YOUR ANSWER?ARTICLE
THAT'S IT!

THE CAT SCRATCHED MIKE.

YOUR ANSWER?SUBJECT
VERY GOOD!

THE CAT SCRATCHED MIKE.

YOUR ANSWER?VERB
WELL DONE!

THE CAT SCRATCHED MIKE

YOUR ANSWER?DIRECT OBJECT
VERY GOOD!

THAT'S ALL FOR TODAY...

YOU GOT 4 CORRECT ANSWER(S).
```

6.4.4 PROGRAM 15: Random Generation of Question Parameters

For drill programs using numerical values, the RND(1) function may be used to ensure that no two RUNs of the program are identical. That is, although the text of the problem may be the same, the parameters are randomly generated in order that each problem appears unique. As an example, consider PROGRAM 15, which gives the base and height of a triangle and asks for the area (Area = 1/2 base × height).

Note: In referring to the listing of PROGRAM 15, look closely at statements 670, 680, and 730. The Apple (and most other brands of microcomputer) will allow multiple BASIC statements per line. This means that a series of statements

may be entered on one line. Each statement is delimited (separated) by a colon. From the system's standpoint, this makes more efficient use of the available memory. From the reader's standpoint, however, it is sometimes more difficult to follow the program's sequence of execution.

In this particular program example, the multiple statements are used to illustrate control of the screen display. If a user's answer is correct, the screen is first erased, a positive response is shown in the center of the screen, and the computer pauses for about 3 seconds before continuing in the instructional sequence.

RUN from disk and refer to the listing and run of PROGRAM 15.

```
]LOAD PROGRAM 15
]LIST

10  REM    PROGRAM 15
20  REM  ==============
30  REM  DRILL AND PRACTICE: THIS
40  REM  PROGRAM DEMOS USER CONTROL OF THE
50  REM  NUMBER OF QUESTIONS TO BE ASKED
60  REM  AND USE OF RND(1) TO RANDOMLY
70  REM  GENERATE NUMBERS WITHIN LIMITS TO
80  REM  USE IN THE TEXT OF A QUESTION.
90  REM  PROGRAM ALSO DEMOS FOR THE FIRST TIME
100  REM  THE USE OF MULTIPLE STATEMENTS PER LINE
110  REM  ==============
120  REM  VARIABLE DICTIONARY
130  REM  ==============
140  REM  A - AREA OF TRIANGLE (ANSWER)
150  REM  B - RANDOMLY SELECTED VALUE FOR
160  REM       THE 'BASE' OF A TRIANGLE
170  REM  H - RANDOMLY SELECTED VALUE FOR
180  REM       THE 'HEIGHT' OF A TRIANGLE
190  REM  P - NUMBER OF PROBLEMS SELECTED BY USER
200  REM  S - USER ANSWER (INPUT)
210  REM  ==============
220  REM
230  HOME
240  PRINT "DRILL ON CALCULATING THE"
250  PRINT "AREA OF A TRIANGLE"
260  PRINT
270  PRINT "HOW MANY PROBLEMS DO YOU WANT";
280  INPUT P
290  IF P < 1 THEN 270
300  IF P < 11 THEN 350
310  PRINT "THAT'S TOO MANY...KEEP"
320  PRINT "    IT TO 10 OR LESS."
330  GOTO 270
340  REM  ===USER GOT TO DEFINE THE VALUE OF P
     (WITHIN LIMITS)===
```

```
350  FOR I = 1 TO P
360  PRINT
370  REM  ===============
380  REM  GET RANDOM VALUES FOR THE BASE AND
390  REM  HEIGHT AND CALCULATE THE AREA.
400  REM  ===============
410 B =  INT (10 *  RND (1) + 1) * 5
420 H =  INT (15 *  RND (1) + 1) * 10
430 A = .5 * B * H
440  PRINT "THE BASE OF A TRIANGLE IS"
450  PRINT B;" INCHES AND ITS HEIGHT"
460  PRINT "IS ";H;" INCHES. WHAT IS"
470  PRINT "ITS AREA IN SQUARE INCHES";
480  INPUT S
490  REM  ===IS INPUT CORRECT ANSWER?===
500  IF A = S THEN 670
510  PRINT
520  PRINT "NO, AREA = 1/2 X BASE X HEIGHT"
530  PRINT "= 1/2 X ";B;" X ";H
540  PRINT "= ";A;" SQUARE INCHES"
550  REM  ===USER CONTROLS WHEN TO GO===
560  PRINT "READY";
570  INPUT Z$
580  GOTO 710
590  REM  =============
600  REM  NOW WE'LL USE MULTIPLE STATEMENTS
610  REM  ON A LINE TO: 1. ERASE THE SCREEN;
620  REM  2. SKIP DOWN TO THE MIDDLE OF THE
630  REM  SCREEN; 3. PRINT A RESPONSE; AND
640  REM  4. HOLD THE DISPLAY WHILE THE
650  REM  SYSTEM "COUNTS" TO 1000.
660  REM  ===============
670  HOME : FOR J = 1 TO 11: PRINT : NEXT J
680  PRINT "      P E R F E C T!": FOR J = 1 TO 1000:
     NEXT J
690 C = C + 1
700  REM  ===CLEAR THE SCREEN THEN GO===
710  HOME
720  NEXT I
730  FOR J = 1 TO 11: PRINT : NEXT J
740  PRINT "Y O U  G O T  ";C;"  C O R R E C T !"
750  END

]RUN
```

[Clear screen]

```
DRILL ON CALCULATING THE
AREA OF A TRIANGLE
```

```
HOW MANY PROBLEMS DO YOU WANT?3

THE BASE OF A TRIANGLE IS
45 INCHES AND ITS HEIGHT
IS 120 INCHES. WHAT IS
ITS AREA IN SQUARE INCHES?2400

NO, AREA = 1/2 BASE X HEIGHT
= 1/2 X 45 X 120
= 2700 SQUARE INCHES
READY?OK
```

[Clear screen]

```
THE BASE OF A TRIANGLE IS
15 INCHES AND ITS HEIGHT
IS 80 INCHES.  WHAT IS
ITS AREA IN SQUARE INCHES?600
```

[Clear screen]

```
      P E R F E C T!
```

[Clear screen]

```
THE BASE OF A TRIANGLE IS
25 INCHES AND ITS HEIGHT
IS 30 INCHES.  WHAT IS
ITS AREA IN SQUARE INCHES?375
```

[Clear screen]

```
      P E R F E C T!
```

[Clear screen]

```
Y O U  G O T  2  C O R R E C T !
```

6.5 TUTORIAL (DIALOG) APPLICATIONS

An extension of the drill-and-practice application allows for more feedback to the user whenever difficulty is indicated. This "tutorial dialog" could assist the user in locating the specific cause of errors, provide hints, or if needed, branch to a separate section for detailed review.

From an instructional computing standpoint, programs of this type are often the most complicated to design, are time-consuming in development, and generally go through many stages of testing and revision. The reason is that these programs (if carefully and thoroughly designed) must anticipate a variety of

users' responses and treat them accordingly: Is the user's answer *partly* correct? Has the user indicated difficulty to the extent that a branch for review is needed? If the user stops in the middle of an interaction, will the program start again at that point for the user? Should the program record the questions/responses for questions missed? Because of these extensive design, development, and evaluation considerations, thorough tutorial dialog programs are not widely available.

The examples that follow are relatively short programs that illustrate some programming strategies for introducing more of a "dialog" into the interaction. They are by no means examples of extensive tutorial dialog instructional computing programs. However, they do show some of the techniques that may be used in programs of this type.

6.5.1 PROGRAM 16: Providing Hints

Providing hints is a simple example of a tutorial program. These hints may be incorporated as DATA elements and READ into a one-dimensional array in the same fashion as questions and answers were in PROGRAM 13. In the following program, a flag is set so that (arbitrarily) a hint is given on the first miss and the correct answer is given on the second miss (see statements 930–980). PROGRAM 16 illustrates this and also follows the criteria defined in Problem 4 of Chapter 5.

RUN from disk and refer to the listing and run of PROGRAM 16.

```
]LOAD PROGRAM 16
]LIST

10  REM   PROGRAM 16
20  REM  ===============
30  REM  TUTORIAL "DIALOG": THIS
40  REM  PROGRAM DEMOS RANDOM QUESTIONS
50  REM  WITH ONE HINT. SELECTION IS FROM
60  REM  ONE-DIM ARRAYS.  DATA IS STORED
70  REM  IN THE SEQUENCE:
80  REM  QUESTION, ANSWER, HINT,
90  REM  FEEDBACK FOR CORRECT AND INCORRECT
100  REM  USER RESPONSES IS ALSO RANDOMLY
110  REM  SELECTED FROM ONE-DIM ARRAYS.
120  REM  ===============
130  REM  VARIABLE DICTIONARY
140  REM  ===============
150  REM  A$( ) - ANSWER FOR QUESTION Q$( )
160  REM  C - NUMBER-CORRECT COUNTER
170  REM  C$( ) - FEEDBACK FOR CORRECT RESPONSE
180  REM  F - FLAG FOR MISSING QUES 1ST TRY
190  REM  H$( ) - HINT FOR QUESTION Q$( )
200  REM  Q - COUNTER FOR NUMBER OF QUESTIONS (LOOP)
210  REM  Q$( ) - QUESTION ASKED
```

```
220  REM  R$ - USER RESPONSE (VIA INPUT)
230  REM  W$( ) - FEEDBACK FOR 1ST INCORRECT RESPONSE
240  REM  X - RANDOM NUMBER (15-1)
250  REM  Z(X) - FLAG FOR RANDOM NUMBER X
260  REM  ===============
270  DIM Q$(15),A$(15),H$(15),Z(15),C$(3),W$(3)
280  REM  ===QUESTION, ANSWER, HINT DATA===
290  DATA  "CAPITAL OF TEXAS","AUSTIN"
300  DATA  "OL' STEPHEN F."
310  DATA  "LAST NAME OF 'BOLERO' COMPOSER"
320  DATA  "RAVEL","SWEATERS CAN UN-"
330  DATA  "FORMULA FOR POTASSIUM FLUORIDE"
340  DATA  "KF","POTASSIUM IS K"
350  DATA  "GOLIATH'S SLAYER","DAVID"
360  DATA  "SLING-SHOT"
370  DATA  "DANIEL WAS PLACED IN THE LION'S -?-"
380  DATA  "DEN","FAMILY ROOM"
390  DATA  "THE 'B' IN BASIC"
400  DATA  "BEGINNER'S","NOVICE'S"
410  DATA  "LARGEST CITY IN JAPAN"
420  DATA  "TOKYO","TOKEN QUESTION"
430  DATA  "A SYNONYM FOR PARONOMASIA"
440  DATA  "PUN","PUNSTERS USE THESE"
450  DATA  "DIVIDE 50 BY 1/2 AND ADD 3. ANS"
460  DATA  "103","((50/.5)+3)"
470  DATA  "GEORGE WASHINGTON COULD NOT TELL A --?--"
480  DATA  "LIE","LITTLE WHITE -?-"
490  DATA  "THREE-TOED SLOTH","AI"
500  DATA  "FIRST AND THIRD VOWELS"
510  DATA  "COOLED LAVA","AA"
520  DATA  "SOUND OF PLEASURE"
530  DATA  "MONTH OF THE WINTER SOLSTICE"
540  DATA  "DECEMBER","MAKE MERRY"
550  DATA  "ADAM'S ALE","WATER","H2O"
560  DATA  "CM PER INCH","2.54","?.54"
570  REM  ===FEEDBACK FOR CORRECT/INCORRECT RESPONSE===
580  DATA  "HOT-DOGGIES","WHOOPS"
590  DATA  "PERFECT","YOU'RE KIDDING"
600  DATA  "SENSATIONAL","THINK OF THIS"
610  REM  ===============
620  REM  STORE QUESTION, ANSWER, HINT
630  REM  ===============
640  FOR I = 1 TO 15
650  READ Q$(I),A$(I),H$(I)
660 Z(I) = 0
670  NEXT I
680  REM  ===============
690  REM  STORE CORRECT, INCORRECT FEEDBACK
700  REM  ===============
710  FOR I = 1 TO 3
```

```
720  READ C$(I),W$(I)
730  NEXT I
740  HOME
750  PRINT "     FUN AND GAMES"
760  PRINT
770  PRINT "HERE ARE 8 QUESTIONS..."
780  REM  ===============
790  REM  ASK 8 QUESTIONS
800  REM  ===============
810  FOR Q = 1 TO 8
820  PRINT
830 F = 0
840 X =  INT (15 *  RND (1) + 1)
850  REM  ===HAS X APPEARED BEFORE?===
860  IF Z(X) = 1 THEN 840
870 Z(X) = 1
880  PRINT Q$(X);
890  INPUT R$
900  REM  ===GET A RANDOM NUMBER FOR FEEDBACK===
910 R =  INT (3 *  RND (1) + 1)
920  IF R$ = A$(X) THEN 990
930  REM  ===HAS QUESTION BEEN MISSED BEFORE?===
940  IF F = 1 THEN 1020
950 F = 1
960  PRINT W$(R);"! HERE'S A HINT:"
970  PRINT H$(X)
980  GOTO 880
990  PRINT C$(R);"!"
1000 C = C + 1
1010  GOTO 1030
1020  PRINT "A CORRECT ANSWER IS ";A$(X)
1030  NEXT Q
1040  PRINT
1050  PRINT "YOU ANSWERED ";C;" CORRECTLY!"
1060  PRINT  TAB( 7);"BYE-BYE..."
1070  END

]RUN
```

[Clear screen]

```
     FUN AND GAMES

HERE ARE 8 QUESTIONS...

LAST NAME OF 'BOLERO' COMPOSER?BACH
WHOOPS! HERE'S A HINT:
SWEATERS CAN UN-
LAST NAME OF 'BOLERO' COMPOSER?RAVEL
PERFECT!
```

```
COOLED LAVA?WHAT?????
YOU'RE KIDDING! HERE'S A HINT:
SOUND OF PLEASURE
COOLED LAVA?AH
A CORRECT ANSWER IS AA

GOLIATH'S SLAYER?DANIEL
WHOOPS! HERE'S A HINT:
SLING-SHOT
GOLIATH'S SLAYER?DAVID
HOT-DOGGIES!

THREE-TOED SLOTH?WHO KNOWS?
YOU'RE KIDDING! HERE'S A HINT:
FIRST AND THIRD VOWELS
THREE-TOED SLOTH?AI
PERFECT!

LARGEST CITY IN JAPAN?TOKYO
HOT-DOGGIES!

THE 'B' IN BASIC?BEGINNERS
YOU'RE KIDDING! HERE'S A HINT:
NOVICE'S
THE 'B' IN BASIC?BEGINNER'S
PERFECT!

GEORGE WASHINGTON COULD NOT TELL A --?--?FIB
THINK OF THIS! HERE'S A HINT:
LITTLE WHITE -?-
GEORGE WASHINGTON COULD NOT TELL A --?--?STORY
A CORRECT ANSWER IS LIE

A SYNONYM FOR PARONOMASIA?PUN
SENSATIONAL!

YOU ANSWERED 6 CORRECTLY!
     BYE-BYE...
```

6.5.2 PROGRAM 17: Review of Missed Questions

A simple expansion of PROGRAM 16 can be accomplished by defining two additional one-dimensional arrays. One will flag a question number missed, and the other will contain the incorrect response entered for that question (see statements 1200–1270). At the conclusion of the program, a list of missed questions, including the incorrect responses and a correct answer, can be displayed for review by the user (see statements 1640–1850).

RUN from disk and refer to the listing and run of PROGRAM 17.

```
]LOAD PROGRAM 17
]LIST

10  REM   PROGRAM 17
20  REM  ===============
30  REM  TUTORIAL "DIALOG": THIS
40  REM  PROGRAM IS SIMILAR TO PROGRAM 16
50  REM  BUT ADDS THE FEATURES OF IMPROVED
60  REM  SCREEN DISPLAY AND "REVIEWS" EACH
70  REM  QUESTION MISSED AT LEAST ONCE.  USER
80  REM  IS ALLOWED TO SELECT THE NUMBER OF
90  REM  QUESTIONS TO BE ASKED.
100  REM  ONLY THOSE QUESTIONS ANSWERED CORRECTLY
110  REM  ON FIRST ATTEMPT ARE COUNTED AS OK.
120  REM  ===============
130  REM  VARIABLE DICTIONARY
140  REM  ===============
150  REM  A$( ) - ANSWER TO QUESTION Q$( )
160  REM  C - NUMBER-CORRECT-1ST-TRY COUNTER
170  REM  C$( ) - FEEDBACK FOR CORRECT RESPONSE
180  REM  F - FLAG FOR MISSING QUESTION 1ST TRY
190  REM  H$( ) - HINT FOR QUESTION Q$( )
200  REM  Q - NUMBER OF QUESTIONS ASKED (VIA USER INPUT)
210  REM  Q$( ) - QUESTION ASKED
220  REM  R - RANDOM NUMBER (3-1) FOR FEEDBACK
230  REM  R$ - USER RESPONSE (VIA INPUT)
240  REM  S$( ) - USER INCORRECT RESPONSE TO QUESTION Q$( )
250  REM  W$( ) - FEEDBACK FOR 1ST INCORRECT RESPONSE
260  REM  X - RANDOM NUMBER (15-1)
270  REM  Z(X) - FLAG FOR RANDOM NUMBER X
280  REM  Z1(X) - FLAG FOR QUESTION NUMBER MISSED BY USER
290  REM  ===============
300  DIM Q$(15),A$(15),H$(15),S$(15),Z(15),Z1(15),C$(3),
     W$(3)
310  REM  ===QUESTION, ANSWER, HINT DATA===
320  DATA  "LATIN FOR 'BUTTOCKS'"
330  DATA  "GLUTEUS MAXIMUS","GLUTEALS"
340  DATA  "LAST NAME OF 'BOLERO' COMPOSER"
350  DATA  "RAVEL","SWEATERS CAN UN-"
360  DATA  "FORMULA FOR ZINC OXIDE"
370  DATA  "ZNO","ZINC IS ZN"
380  DATA  "GOLIATH'S SLAYER","DAVID"
390  DATA  "SLINGSHOT"
400  DATA  "DANIEL WAS PLACED IN THE LION'S -?-"
410  DATA  "DEN","FAMILY ROOM"
420  DATA  "WHAT KEEPS THE DOCTOR AWAY"
430  DATA  "AN APPLE A DAY","MICROCOMPUTER"
440  DATA  "WHICH IS LONGER: METER OR YARD"
```

```
450  DATA  "METER","A METER IS 39.37 IN."
460  DATA  "MONTH OF THE LONGEST DAY"
470  DATA  "JUNE","ASSUME N. HEMISPHERE"
480  DATA  "THE 'B' IN BASIC"
490  DATA  "BEGINNER'S","NOVICE'S"
500  DATA  "A THREE-TOED SLOTH","AI"
510  DATA  "FIRST AND THIRD VOWELS"
520  DATA  "LARGEST RIVER IN THE WORLD"
530  DATA  "AMAZON","BIG-MOMMA!"
540  DATA  "LONGEST RIVER IN THE WORLD"
550  DATA  "NILE","A SHADE OF GREEN"
560  DATA  "NORTHERN WATER BODY OF TEXAS"
570  DATA  "RED RIVER","RED RYDER"
580  DATA  "CAPITAL OF OREGON","SALEM"
590  DATA  "WITCH HUNT CITY"
600  DATA  "'NEW DOOR' REARRANGED TO ONE WORD"
610  DATA  "ONE WORD","IT'S ONE WORD"
620  REM  ===FEEDBACK FOR CORRECT/INCORRECT RESPONSE===
630  DATA  "GREAT","OH, SHOOT"
640  DATA  "SENSATIONAL","WHOA NOW..."
650  DATA  "HOT-DOGGIES","LET ME HELP"
660  REM  ===============
670  REM  STORE QUESTION, ANSWER, HINT
680  REM  ===============
690  FOR I = 1 TO 15
700  READ Q$(I),A$(I),H$(I)
710 Z(I) = 0:Z1(I) = 0
720  NEXT I
730  REM  ===============
740  REM  STORE FEEDBACK FOR CORRECT/INCORRECT RESPONSE===
750  FOR I = 1 TO 3
760  READ C$(I),W$(I)
770  NEXT I
780  HOME : PRINT : PRINT : PRINT
790  PRINT "MORE FUN AND GAMES": PRINT
800  PRINT
810  PRINT "WHAT'S YOUR FIRST NAME";
820  INPUT F$
830  HOME : PRINT
840  PRINT
850  REM  ===INTRODUCTION===
860  PRINT "HELLO, ";F$;".  THIS PROGRAM": PRINT
870  PRINT "SHOWS SOME SIMPLE USES OF BASIC": PRINT
880  PRINT "IN INSTRUCTIONAL COMPUTING. WHAT": PRINT
890  PRINT "I'LL DO IS ASK SOME QUESTIONS": PRINT
900  PRINT "THAT ARE 'FUN AND GAMES.' IF YOU": PRINT
910  PRINT "MISS ON THE FIRST TRY, I'LL GIVE": PRINT
920  PRINT "A HINT. I'LL ONLY GIVE YOU TWO": PRINT
930  PRINT "CHANCES TO ANSWER. AT THE END,": PRINT
940  PRINT "I'LL SHOW YOU THE QUESTIONS MISSED": PRINT
```

```
950  PRINT "AND GIVE YOU YOUR SCORE."
955  REM === INPUT STATEMENT CAN HAVE TEXT ===
960  PRINT : INPUT "READY?";Z$
970  REM  ===CLEAR AND CENTER ON SCREEN===
980  HOME : FOR I = 1 TO 11: PRINT : NEXT I
990  PRINT "I HAVE 15 QUESTIONS AVAILABLE."
1000  PRINT "HOW MANY WOULD YOU LIKE";: INPUT Q
1010  REM  ===CHECK FOR WITHIN RANGE===
1020  IF Q < 1 THEN 1050
1030  IF Q > 15 THEN 1050
1040  GOTO 1080
1050  PRINT "AWWW, IT HAS TO BE 1 TO 15!"
1060  GOTO 990
1070  REM  ===ASK "Q" QUESTIONS===
1080  FOR J = 1 TO Q
1090  HOME : FOR I = 1 TO 10: PRINT : NEXT I
1100 F = 0
1110 X =  INT (15 *  RND (1) + 1)
1120  REM  ===HAS X APPEARED BEFORE?===
1130  IF Z(X) = 1 THEN 1110
1140 Z(X) = 1
1150  PRINT Q$(X);
1160  INPUT R$
1170 R =  INT (3 *  RND (1) + 1)
1180  IF R$ = A$(X) THEN 1360
1190  IF F = 1 THEN 1470
1200  REM  ===============
1210  REM  FOR A QUESTION THAT IS MISSED:
1220  REM  1. FLAG THE NUMBER OF THAT QUESTION;
1230  REM  2. STORE THE RESPONSE GIVEN BY THE USER
1240  REM      IN S$(X) FOR LATER RECALL.
1250  REM  ===============
1260 Z1(X) = 1
1270 S$(X) = R$
1280 F = 1
1290  PRINT W$(R);"! HERE'S A HINT:"
1300  PRINT H$(X)
1310  GOTO 1150
1320  REM  ===============
1330  REM  CLEAR THE SCREEN, CENTER THE FEEDBACK,
1340  REM  AND HOLD IT THERE A MOMENT OR TWO.
1350  REM  ================
1360  HOME : FOR I = 1 TO 11: PRINT : NEXT I
1370  PRINT  TAB( 10);C$(R);"!"
1380  FOR I = 1 TO 1000: NEXT I
1390  REM  ===NO CREDIT GIVEN IF CORRECT 2ND
      TRY!===
1400  IF F = 1 THEN 1500
1410 C = C + 1
```

```
1420  GOTO 1500
1430  REM  ===============
1440  REM  FLASH THE CORRECT ANSWER AND
1450  REM  LET USER SAY WHEN TO GO.
1460  REM  ===============
1470  PRINT "A CORRECT ANSWER IS: ";
1480  FLASH : PRINT A$(X): NORMAL
1490  PRINT : INPUT "READY?";Z$
1500  NEXT J
1510  HOME : PRINT
1520  PRINT "YOU ANSWERED ";C;" CORRECTLY"
1530  PRINT "ON THE FIRST TRY..."
1540  REM  ===EVERY ANSWER CORRECT FIRST TRY?===
1550  IF C <  > Q THEN 1600
1560  PRINT
1570  PRINT "E X C E L L E N T !"
1580  FOR I = 1 TO 3000: NEXT I
1590  GOTO 1860
1600  PRINT : PRINT
1610  PRINT "THE QUESTIONS MISSED AT"
1620  PRINT "LEAST ONCE ARE:"
1630  FOR I = 1 TO 5000: NEXT I
1640  REM  ===============
1650  REM  LOOP THRU ALL 15 TO SEE IF A QUESTION
1660  REM  WAS MISSED.  IF MISSED, Z1(I) IS 1
1670  REM   (FROM SETTING THE FLAG AT LINE 1260)
1680  REM  AND THE USER'S INCORRECT RESPONSE
1690  REM   IS STORED IN S$(I). (FROM LINE 1270)
1700  REM  ===============
1710  FOR I = 1 TO 15
1720  HOME : PRINT : PRINT : PRINT
1730  REM  ===DID THE USER MISS THIS QUESTION?===
1740  IF Z1(I) = 0 THEN 1850
1750  PRINT
1760  PRINT "     QUESTION:"
1770  PRINT Q$(I)
1780  PRINT
1790  PRINT "     YOUR ANSWER:"
1800  PRINT S$(I)
1810  PRINT
1820  PRINT "     A CORRECT ANSWER:"
1830  PRINT A$(I): PRINT
1840  INPUT "     READY?";Z$
1850  NEXT I
1860  HOME : FOR I = 1 TO 10: PRINT : NEXT I
1870  REM  ===COMPUTE A PERCENTAGE SCORE===
1880  PRINT "YOUR SCORE IS ";C * 100 / Q;" %!"
1890  PRINT  TAB( 5);"BYE-BYE, ";F$
1900  END
```

```
]RUN
```

[Clear screen]

```
MORE FUN AND GAMES

WHAT'S YOUR FIRST NAME?SAMMY
```

[Clear screen]

```
HELLO, SAMMY.  THIS PROGRAM

SHOWS SOME SIMPLE USES OF BASIC

IN INSTRUCTIONAL COMPUTING. WHAT

I'LL DO IS ASK SOME QUESTIONS

THAT ARE 'FUN AND GAMES.' IF YOU

MISS ON THE FIRST TRY, I'LL GIVE

A HINT.  I'LL ONLY GIVE YOU TWO

CHANCES TO ANSWER. AT THE END,

I'LL SHOW YOU THE QUESTIONS MISSED

AND GIVE YOU YOUR SCORE.

READY?OK
```

[Clear screen]

```
I HAVE 15 QUESTIONS AVAILABLE.
HOW MANY WOULD YOU LIKE?0
AWWW, IT HAS TO BE 1 TO 15!
I HAVE 15 QUESTIONS AVAILABLE.
HOW MANY WOULD YOU LIKE?100
AWWW, IT HAS TO BE 1 TO 15!
I HAVE 15 QUESTIONS AVAILABLE.
HOW MANY WOULD YOU LIKE?3
```

[Clear screen]

```
LONGEST RIVER IN THE WORLD?MISSISSIPPI
WHOA NOW...! HERE'S A HINT:
A SHADE OF GREEN
LONGEST RIVER IN THE WORLD?THAMES
A CORRECT ANSWER IS: NILE

READY?YES
```

[Clear screen]

```
LARGEST RIVER IN THE WORLD?AMAZON
```

[Clear screen]

```
          HOT-DOGGIES!
```

[Clear screen]

```
A THREE-TOED SLOTH?WHO KNOWS?
LET ME HELP! HERE'S A HINT:
FIRST AND THIRD VOWELS
A THREE-TOED SLOTH?AI
```

[Clear screen]

```
          GREAT!
```

[Clear screen]

```
YOU ANSWERED 1 CORRECTLY
ON THE FIRST TRY...

THE QUESTIONS MISSED AT
LEAST ONCE ARE:
```

[Clear screen]

```
     QUESTION:
A THREE-TOED SLOTH

     YOUR ANSWER:
WHO KNOWS?

     A CORRECT ANSWER:
AI

     READY?YES

     QUESTION:
LONGEST RIVER IN THE WORLD

     YOUR ANSWER:
MISSISSIPPI

     A CORRECT ANSWER:
NILE

     READY?GO ON...
```

[Clear screen]

```
YOUR SCORE IS 33.3333333 %!
    BYE-BYE, SAMMY
```

6.5.3 PROGRAM 18: Model Tutorial Program with Hints

Based upon the previous two programs, it is possible to derive a "model" program for a general type of tutorial interaction. For this model, a maximum of 50 possible questions and the presentation of two hints have been arbitrarily defined. The program is designed so that DATA elements corresponding to the number of possible questions, the number of questions to be presented, the text of the question, the text of the answer, and the text of the first and second hints are added to the program following statement 1670 (see statements 1590–1670).

This model program shows that, once a general design has been defined, it is a relatively simple task to use the same program in a variety of conceptual applications. By adding an appropriate introduction via PRINT statements and DATA containing whatever content is desired, the program will ask questions, provide hints, and so forth, for any chosen topic.

Refer to the listing of PROGRAM 18.

```
]LOAD PROGRAM 18
]LIST

10  REM   PROGRAM 18
20  REM  ===============
30  REM  TUTORIAL "DIALOG": THIS PROGRAM
40  REM  IS BASED ON PROGRAM 17 CONCEPTS.
50  REM  PROGRAM DEMOS HOW "MODELS" MAY BE
60  REM  BUILT SO THAT DATA CORRESPONDING TO
70  REM  QUESTIONS, ANSWERS, AND HINTS OF A
80  REM  USER'S CHOOSING MAY BE ADDED TO THE
90  REM  MODEL PROGRAM.  THIS PROGRAM
100  REM  ARBITRARILY HAS FIVE CHOICES FOR
110  REM  CORRECT/INCORRECT FEEDBACK INCLUDED.
120  REM  UP TO TWO HINTS ARE GIVEN FOR EACH
130  REM  INCORRECT RESPONSE.  REVIEW OF MISSED
140  REM  QUESTIONS IS GIVEN AT CONCLUSION OF
150  REM  THE PROGRAM.  DATA REPRESENTING THE
160  REM  NUMBER OF QUESTIONS AVAILABLE (N1)
170  REM  AND NUMBER OF QUESTIONS TO BE ASKED (N2)
180  REM  PLUS DATA FOR QUESTION, ANSWER, HINT1,
190  REM  AND HINT2 ARE ADDED AFTER LINE 1670.
200  REM  ===============
210  REM  VARIABLE DICTIONARY
220  REM  ===============
230  REM  A$( ) - CORRECT ANSWER
240  REM  C - NUMBER-CORRECT COUNTER
250  REM  C$( ) - RANDOM POSITIVE FEEDBACK
260  REM  F - COUNTER FOR THE NUMBER OF TIMES
270  REM          A QUESTION HAS BEEN MISSED
280  REM  H1$( ) - FIRST HINT
290  REM  H2$( ) - SECOND HINT
300  REM  N1 - TOTAL NUMBER OF QUES, ANS, HINTS
```

```
310  REM        TO BE READ FROM DATA
320  REM  N2 - NUMBER OF QUESTIONS TO ASK IN LOOP
330  REM  Q$( ) - QUESTION ASKED
340  REM  R$ - USER RESPONSE (VIA INPUT)
350  REM  S$( ) - USER'S INCORRECT RESPONSE
360  REM  W$( ) - RANDOM FEEDBACK FOR INCORRECT RESPONSE
370  REM  X - RANDOM NUMBER (N1 TO 1)
380  REM  Z(X) - RANDOM NUMBER SELECTION FLAG
390  REM  Z1(X) - FLAG FOR THE QUES. NO. MISSED
400  REM  ==============
410  DIM Q$(50),A$(50),H1$(50),H2$(50)
420  DIM S$(50),Z(50),Z1(50),C$(5),W$(5)
430  REM  ===FEEDBACK FOR CORRECT/INCORRECT RESPONSE===
440  DATA  "GRRREAT","NO...THINK OF THIS"
450  DATA  "FINE","HOLD IT","PERFECT","NO...NOT YET"
460  DATA  "HOT-DOG","LET ME HELP"
470  DATA  "MARVELOUS","THIS MAY HELP"
480  REM  ===STORE CORRECT, INCORRECT RESPONSES===
490  FOR I = 1 TO 5
500  READ C$(I),W$(I)
510  NEXT I
520  REM  ===============
530  REM  READ THE NUMBER OF QUESTION "SETS"
540  REM  AVAILABLE FROM DATA AND THE NUMBER
550  REM  OF QUESTIONS TO ASK IN THE LOOP.
560  REM  ================
570  READ N1,N2
580  REM  ===NOW STORE QUES, ANS, HINT1, HINT2===
590  FOR I = 1 TO N1
600  READ Q$(I),A$(I),H1$(I),H2$(I)
610 Z(I) = 0
620 Z1(I) = 0
630  NEXT I
640 C = 0
650  REM  ==============
660  REM  INTRODUCTORY STATEMENTS MAY BE ADDED
670  REM   HERE UP TO LINE NUMBER 1000.
680  REM  ===============
1000  FOR Q = 1 TO N2
1010  HOME :F = 0
1020  PRINT : PRINT : PRINT
1030 X =  INT (N1 *  RND (1) + 1)
1040  IF Z(X) = 1 THEN 1030
1050 Z(X) = 1
1060  REM  ===ASK THE QUESTION===
1070  PRINT : PRINT Q$(X);
1080  INPUT R$
1090 R =  INT (5 *  RND (1) + 1)
1100  IF R$ = A$(X) THEN 1350
1110  HOME : FOR I = 1 TO 10: PRINT : NEXT I
```

```
1120 REM  ===FLAG QUES. NO. MISSED AND STORE RESPONSE===
1130 Z1(X) = 1
1140 S$(X) = R$
1150 F = F + 1
1160  REM  ===============
1170  REM  GIVE EITHER THE FIRST HINT, THE SECOND
1180  REM  HINT, OR THE CORRECT ANSWER.
1190  REM  ==============
1200  ON F GOTO 1210,1240,1270
1210  PRINT W$(R);"! HERE'S A HINT:"
1220  PRINT H1$(X)
1230  GOTO 1070
1240  PRINT W$(R);"! HERE'S ANOTHER HINT:"
1250  PRINT H2$(X)
1260  GOTO 1070
1270  PRINT "A CORRECT ANSWER IS ";A$(X)
1280  PRINT "READY TO GO ON";
1290  INPUT Z$
1300  GOTO 1400
1310  REM  ===============
1320  REM  CLEAR, CENTER, GIVE FEEDBACK
1330  REM  AND HOLD IT FOR A MOMENT.
1340  REM  ===============
1350  HOME : FOR I = 1 TO 11: PRINT : NEXT I
1360  PRINT  TAB( 10);C$(R);"!"
1370  FOR I = 1 TO 1000: NEXT I
1380  IF F <  > 0 THEN 1400
1390 C = C + 1
1400  NEXT Q
1410  HOME : FOR I = 1 TO 10: PRINT : NEXT I
1420  PRINT "FIRST TRY CORRECT = ";C
1430  IF C <  > N2 THEN 1460
1440  PRINT "E X C E L L E N T !"
1450  GOTO 9999
1460  PRINT "THE ONES MISSED AT LEAST ONCE ARE:"
1470  FOR I = 1 TO N1
1480  IF Z1(I) = 0 THEN 1580
1490  PRINT : PRINT : PRINT
1500  PRINT "QUESTION: ";Q$(I)
1510  PRINT
1520  PRINT "YOUR ANSWER: ";S$(I)
1530  PRINT
1540  PRINT "CORRECT ANSWER: ";A$(I)
1550  PRINT : PRINT
1560  PRINT  TAB( 10);"READY";
1570  INPUT Z$
1580  NEXT I
1590  REM  ===============
1600  REM  ADD A DATA STATEMENT HERE FOR
1610  REM  THE NUMBER OF QUESTION "SETS" AND
```

```
1620  REM  THE NUMBER OF QUESTIONS TO BE ASKED.
1630  REM  EXAMPLE: DATA 25,15
1640  REM  THEN ADD DATA FOR THE QUESTION "SETS"
1650  REM  IN THE SEQUENCE "QUESTION", "ANSWER"
1660  REM  "FIRST HINT","SECOND HINT"
1670  REM  ============
9999  PRINT "   BYE-BYE..."
10000  END
```

6.5.4 PROGRAM 19: Model Tutorial Program with Dialog

A tutorial program can do more than just give hints when users are having difficulty with a given question: It can, to some degree, approach the type of dialog that occurs between a tutor and a student. As an example, consider a question related to the chemical concept of a mole. By definition, a *mole* is a quantity of a chemical compound equal to the formula weight (FW) of that compound. This quantity is usually expressed in grams, but it could be any mass unit (ounces, tons, and so on). For a given weight of a chemical compound, the number of moles is determined by the following formula:

$$\text{Moles} = \text{Weight (grams)/FW (gram-formula weight)}$$

The following program illustrates a type of dialog that could occur in a tutorial instructional computing application. Note that the program makes use of the ABS (absolute) function to allow for a tolerance of ± 0.1 in the user's answers (see statements 450–500 and 680). Also be reminded that this is, in essence, a program fragment and does not include an introduction, examples, random selection of positive responses, use of counters for the number correct, and so on. These elements should always be incorporated in programs for actual use in an educational setting.

RUN from disk and refer to the listing and run of PROGRAM 19.

```
]LOAD PROGRAM 19
]LIST

10  REM   PROGRAM 19
20  REM  ==============
30  REM  TUTORIAL "DIALOG": THIS
40  REM  PROGRAM DEMOS MORE OF A "TUTORIAL"
50  REM  TYPE OF INTERACTION BETWEEN USER AND
60  REM  THE PROGRAM, USING THE CHEMICAL CONCEPT
70  REM  OF THE "MOLE" AS AN ILLUSTRATIVE VEHICLE.
80  REM  ONLY THREE COMPOUNDS ARE USED IN
90  REM  THE EXAMPLE WITH THEIR FORMULAS AND
100  REM  FORMULA WEIGHTS STORED IN ONE-DIMENSION ARRAYS.
110  REM  USE OF THE ABS (ABSOLUTE) FUNCTION
120  REM  IS ALSO INTRODUCED.
```

```
130  REM  ============
140  REM  VARIABLE DICTIONARY
150  REM  ============
160  REM  C$( ) - CHEMICAL COMPOUND FORMULA
170  REM  F - FLAG FOR MISSING QUESTION 1ST TRY
180  REM  G - RANDOM NUMBER OF GRAMS OF COMPOUND
190  REM  M - NUMBER OF MOLES (GRAMS/FORMULA WT)
200  REM  R, R$, V - USER RESPONSES (VIA INPUTS)
210  REM  W( ) - FORMULA WEIGHT OF A COMPOUND
220  REM  X - RANDOM NUMBER (3 TO 1) FOR
230  REM       COMPOUND SELECTION
240  REM  ===============
250  DIM C$(3),W(3)
260  DATA  "KOH",56,"HF",20,"KI",166
270  REM  ===STORE THE FORMULAS AND WEIGHTS===
280  FOR I = 1 TO 3
290  READ C$(I),W(I)
300  NEXT I
310 F = 0
320  REM  ===============
330  REM  GET A RANDOM NUMBER OF GRAMS AND A
340  REM  RANDOM COMPOUND. THEN CALCULATE MOLES.
350  REM  ================
360 G =  INT (10 *  RND (1) + 1) * 20
370 X = INT (3 *  RND (1) + 1)
380 M = G / W(X)
390  REM  ===ASK THE QUESTION===
400  PRINT
410  PRINT "HOW MANY MOLES OF ";C$(X);" ARE"
420  PRINT "PRESENT IN ";G;" GRAMS";
430  INPUT R
440  PRINT
450  REM  ===============
460  REM  USE THE ABS FUNCTION TO ACCEPT AN
470  REM  ANSWER THAT IS WITHIN 0.1 OF THE
480  REM  CORRECT ANSWER AND THE USER'S ANSWER.
490  REM  ===============
500  IF  ABS (R - M) <  = .1 THEN 860
510  REM  ===GIVE ANSWER ON SECOND MISS===
520  IF F = 1 THEN 800
530 F = 1
540  REM  ===ASK FIRST STEP IN SOLUTION SEQUENCE===
550  PRINT "NO...DID YOU DIVIDE THE"
560  PRINT "WEIGHT BY THE FW (Y OR N)";
570  INPUT R$
580  PRINT
590  IF R$ = "Y" THEN 630
600  PRINT "WELL, YOU SHOULD! TRY AGAIN."
610  GOTO 400
620  REM  ===CHECK FOR SECOND STEP IN SOL'N SEQUENCE===
```

```
630  PRINT "GOOD. THAT IS CORRECT."
640  PRINT "WHAT VALUE DID YOU USE"
650  PRINT "FOR THE FW OF ";C$(X);
660  INPUT V
670  PRINT
680  IF  ABS (V - W(X)) < = .1 THEN 750
690  PRINT "AHA!  THIS MAY BE YOUR"
700  PRINT "PROBLEM.  THE APPROXIMATE"
710  PRINT "FW OF ";C$(X);" IS ";W(X)
720  PRINT "NOW TRY IT AGAIN..."
730  GOTO 400
740  REM  ===IF CORRECT TO HERE, MUST BE MATH ERROR===
750  PRINT "HMMM...THAT IS THE CORRECT"
760  PRINT "FW FOR ";C$(X);". YOU MUST"
770  PRINT "HAVE MADE AN ARITHMETIC"
780  PRINT "ERROR.  CHECK AND TRY AGAIN."
790  GOTO 400
800  PRINT
810  REM  ===SHOW THE CORRECT SOLUTION===
820  PRINT "MOLES = WT/FW"
830  PRINT "= ";G;"/";W(X)
840  PRINT "= "; INT (M * 100) / 100
850  GOTO 880
860  PRINT "   E X C E L L E N T !"
870  REM  ===LET USER CONTINUE AT WILL===
880  PRINT
890  PRINT "WANT ANOTHER (Y OR N)";
900  INPUT R$
910  IF R$ = "Y" THEN 310
920  END

]RUN

HOW MANY MOLES OF KI ARE
PRESENT IN 200 GRAMS?33

NO...DID YOU DIVIDE THE
WEIGHT BY THE FW (Y OR N)?Y

GOOD. THAT IS CORRECT.
WHAT VALUE DID YOU USE
FOR THE FW OF KI?56

AHA!  THIS MAY BE YOUR
PROBLEM.  THE APPROXIMATE
FW OF KI IS 166
NOW TRY IT AGAIN...

HOW MANY MOLES OF KI ARE
PRESENT IN 200 GRAMS?1.2
```

```
   E X C E L L E N T !

WANT ANOTHER (Y OR N)?Y

HOW MANY MOLES OF HF ARE
PRESENT IN 120 GRAMS?2

NO...DID YOU DIVIDE THE
WEIGHT BY THE FW (Y OR N)?N

WELL, YOU SHOULD! TRY AGAIN.

HOW MANY MOLES OF HF ARE
PRESENT IN 120 GRAMS?1.5

MOLES = WT/FW
= 120/20
= 6

WANT ANOTHER (Y OR N)?Y

HOW MANY MOLES OF KI ARE
PRESENT IN 120 GRAMS?.5

NO...DID YOU DIVIDE THE
WEIGHT BY THE FW (Y OR N)?Y

GOOD. THAT IS CORRECT.
WHAT VALUE DID YOU USE
FOR THE FW OF KI?166

HMMM...THAT IS THE CORRECT
FW FOR KI. YOU MUST
HAVE MADE AN ARITHMETIC
ERROR.  CHECK AND TRY AGAIN.

HOW MANY MOLES OF KI ARE
PRESENT IN 120 GRAMS?.71

   E X C E L L E N T !

WANT ANOTHER (Y OR N)?N
```

6.6 SIMULATION APPLICATIONS

Usually, simulation applications in instructional computing are used when it is important to understand a given concept but one or more of the following situations apply:

1. There are time and/or space and/or equipment limitations.
2. The real process might place the user in a perilous situation.
3. Review and/or practice would be beneficial prior to performing the actual experiment or process.

Simulations in instructional computing are based upon models, most of which are mathematical in origin. In general, these programs allow a user to manipulate parameters and perhaps discover the effect of those manipulations. One popular application is in population dynamics: What happens to the population over a certain span of years if both the birth and death rates decrease and the female/male birth ratio increases? Another application is in environmental studies: What happens to the water oxygen content if untreated raw sewage is dumped into a slow-moving stream? A fast-moving river? What is the effect of performing primary treatment? Secondary treatment? How does temperature affect the above results?

Again, the example programs discussed below are not extensive simulations; they do illustrate the concept of basing the design on some defined model.

6.6.1 PROGRAM 20: Caloric Intake and Ideal Weight

Diet and proper body weight maintenance are popular concerns in our society. It is well known that, by careful control of calorie intake and a good exercise program, weight can be lost or gained and then maintained at an "ideal" level.

PROGRAM 20 is based upon a model which defines a woman's ideal weight as 100 pounds plus (or minus) 5 pounds for each inch over (or under) 5 feet in height. A man's ideal weight is defined as 106 pounds plus (or minus) 6 pounds for each inch over (or under) 5 feet in height. This weight, multiplied by an exercise activity factor of 12 if not active, 15 if moderately active, or 18 if very active, gives an approximate daily calorie count to maintain the ideal weight. A prediction of weight loss or gain can be made by comparing this count with an actual (or, in this case, simulated) daily caloric intake.

PROGRAM 20 allows manipulation of a limited daily menu and exercise activity factors to examine the effects on weight control. However, its use here is primarily that of a simple illustration of basing a simulation on a defined model. Any model (within reason) can be simulated by a computer program. The most important step in developing the program is careful analysis of its design from the model.

RUN from disk and refer to the listing and run of PROGRAM 20.

```
]LOAD PROGRAM 20
]LIST

10  REM  PROGRAM 20 DESCRIPTION
20  REM  ======================
30  REM  SIMULATION:  THIS PROGRAM PRESENTS A
```

```
40  REM  SIMULATED (AND LIMITED!) DAILY MENU
50  REM  FOR SELECTION BY A USER.  BASED UPON
60  REM  THE MENU SELECTED AND THE SEX, HT.,
70  REM  AND ACTIVITY OF THE USER, AN IDEAL
80  REM  WEIGHT AND CALORIC INTAKE TO MAIN-
90  REM  TAIN THAT WEIGHT IS GIVEN.  FINALLY,
100  REM  A WEIGHT DIFFERENTIAL (LOSS OR GAIN)
110  REM  ASSUMING CONSISTENT CALORIC INTAKE
120  REM  IS SHOWN. BUT, THE MENU IS LIMITED!
130  REM  ====================
140  REM
150  REM  VARIABLE DICTIONARY
160  REM  ====================
170  REM  A - EXERCISE ACTIVITY FACTOR
180  REM  B - BASE WEIGHT
190  REM  C - CALORIES TO MAINTAIN IDEAL WT.
200  REM  C() - CALORIES PER FOOD PORTION
210  REM  E - TYPE OF EXERCISE
220  REM  F$() - FOOD LIST FOR MEALS
230  REM  H - HEIGHT IN INCHES
240  REM  I - IDEAL WEIGHT
250  REM  M$ - MEAL
260  REM  N - NUMBER OF THE FOOD SELECTED
270  REM  P - POUNDS (LOSS OR GAIN)
280  REM  S - SEX OF THE USER
290  REM  T - TOTAL CALORIC INTAKE
300  REM  W - WT. FACTOR/INCH FROM 5 FT.
310  REM  Z() - FLAG FOR FOOD SELECTED
320  REM  ====================
330  REM
340  DIM C(11),F$(11),Z(11)
350  HOME : FOR J = 1 TO 9: PRINT : NEXT J
360  PRINT "A SIMULATED DAILY CALORIC INTAKE AND"
370  PRINT : PRINT "  ITS EFFECT ON YOUR IDEAL WEIGHT"
380  FOR J = 1 TO 4000: NEXT J
390  HOME : PRINT : PRINT "YOU WILL BE PRESENTED A MENU FOR BREAK-"
400  PRINT : PRINT "FAST, LUNCH, AND DINNER. SELECT AS MANY"
410  PRINT : PRINT "ITEMS FROM EACH MENU AS YOU WISH. AFTER"
420  PRINT : PRINT "YOUR DAILY MENU HAS BEEN COMPLETED, YOU"
430  PRINT : PRINT "WILL RECEIVE A SUMMARY OF YOUR CALORIC"
440  PRINT : PRINT "INTAKE AND ITS EFFECT ON YOUR IDEAL WT."
450  PRINT : PRINT "     D E P R E S S   A N Y   K E Y..."
460  GET Z$
470  REM
480  REM  ===BREAKFAST DATA===
490  REM
500  DATA  "BREAKFAST","BACON OR SAUSAGE",200,"CEREAL WITH MILK",250
510  DATA  "COFFEE (BLACK)",5,"COFFEE (WITH SUGAR)",50
520  DATA  "EGGS (2)",100,"MILK",125
530  DATA  "ORANGE JUICE",60,"PANCAKES",225
```

```
540  DATA  "SWEET ROLL",250,"TOAST",75,"WAFFLES",550
550  GOSUB 1290
560  REM
570  REM  ===LUNCH DATA===
580  REM
590  DATA  "LUNCH","BEER",125,"BEFORE LUNCH DRINK",115
600  DATA  "CHEESEBURGER",310,"COLA",144
610  DATA  "COTTAGE CHEESE",110,"CRACKERS",75
620  DATA  "FRENCH FRIES",400,"HAMBURGER",260
630  DATA  "MILK",125,"TUNA FISH",50
640  DATA  "VEGETABLE OR FRUIT SALAD",75
650  GOSUB 1290
660  REM
670  REM  ===DINNER DATA===
680  REM
690  DATA  "DINNER","APPLE (OF COURSE) PIE",300
700  DATA  "BAKED POTATO",250,"BEFORE DINNER DRINK",115
710  DATA  "BEEF STEAK",560,"BEETS",40
720  DATA  "DOZEN RAW OYSTERS",240,"FISH",400
730  DATA  "MACARONI",85,"PEAS",115
740  DATA  "TOSSED SALAD",75,"T.V. DINNER",500
750  GOSUB 1290
760  GOSUB 1530
770  PRINT "NOW, SOME PERSONAL DATA IS NEEDED..."
780  FOR J = 1 TO 3000: NEXT J
790  GOSUB 1530
800  PRINT "ARE YOU:"
810  PRINT "   1. FEMALE"
820  PRINT "   2. MALE"
830  PRINT "ENTER 1 OR 2";
840  INPUT S
850  IF S < 1 OR S > 2 THEN 830
860  IF S = 1 THEN B = 100:W = 5: GOTO 880
870 B = 106:W = 6
880  GOSUB 1530
890  PRINT "WHAT IS YOUR HEIGHT IN INCHES";
900  INPUT H
910  IF H < 48 OR H > 84 THEN 890
920 I = ((H - 60) * W) + B
930  GOSUB 1530
940  PRINT "DO YOU CONSIDER YOURSELF:"
950  PRINT "   1. SEDENTARY (LITTLE EXERCISE)"
960  PRINT "   2. MODERATELY ACTIVE"
970  PRINT "   3. VERY ACTIVE"
980  PRINT "ENTER 1, 2, OR 3";
990  INPUT E
1000  IF E < 1 OR E > 3 THEN 980
1010  IF E = 1 THEN A = 12: GOTO 1040
1020  IF E = 2 THEN A = 15: GOTO 1040
1030  IF E = 3 THEN A = 18
```

```
1040 C = I * A
1050  HOME : PRINT TAB(  8);"SUMMARY OF DATA": PRINT
1060  PRINT "YOUR IDEAL WEIGHT IS ";I: PRINT
1070  PRINT "TO MAINTAIN THAT WEIGHT YOU NEED"
1080  PRINT C;" CALORIES PER DAY.": PRINT
1090  PRINT "YOUR DAILY CALORIC INTAKE BASED UPON"
1100  PRINT "THE LIMITED MENU IS ";T;" CALORIES.": PRINT
1110  PRINT "   D E P R E S S  A N Y  K E Y..."
1120  GET Z$
1130  HOME : PRINT : PRINT : PRINT "     D A T A   A N A L Y S I S"
1140 P =  INT ((((T - C) * 7) / 3500) * 10) / 10
1150  FOR J = 1 TO 4: PRINT : NEXT J
1160  PRINT : PRINT "IF YOU ARE CONSISTENT IN THIS CALORIC"
1170  PRINT : PRINT "INTAKE, YOUR WEIGHT DIFFERENTIAL WILL"
1180  PRINT : PRINT "BE APPROXIMATELY ";P;" POUNDS/WEEK."
1190  PRINT : PRINT "   D E P R E S S   A N Y   K E Y..."
1200  GET Z$
1210  GOSUB 1530
1220  PRINT "DO YOU WISH ANOTHER ANALYSIS (Y OR N)";
1230  INPUT Z$
1240  IF Z$ = "Y" THEN T = 0: RESTORE : GOTO 550
1250  GOSUB 1530
1260  PRINT "MAY YOUR BODY BE BEAUTIFUL..."
1270  FOR J = 1 TO 4000: NEXT J: HOME
1280  END
1290  READ M$
1300  FOR J = 1 TO 11: READ F$(J),C(J): NEXT J
1310  HOME : PRINT  TAB( 14);M$: PRINT
1320  PRINT "  FOOD"; TAB( 28);"CALORIES": PRINT
1330  FOR J = 1 TO 11
1340  IF Z(J) = 0 THEN 1390
1350  REM
1360  REM  ===INVERSE ITEMS SELECTED; GIVE CALORIES===
1370  REM
1380  INVERSE : PRINT J;". ";F$(J);: NORMAL : PRINT  TAB( 30);C(J):
      GOTO 1400
1390  PRINT J;". ";F$(J)
1400  NEXT J
1410  PRINT "12. CONTINUE TO NEXT SECTION..."
1420  PRINT : PRINT "YOUR CHOICE(S) (1 TO 12)";
1430  INPUT N
1440  IF N < 1 OR N > 12 THEN 1420
1450  IF N = 12 THEN 1510
1460  REM
1470  REM  ===FLAG THE NUMBER OF THE ITEM SELECTED===
1480  REM
1490 Z(N) = 1
1500 T = T + C(N): GOTO 1310
1510  FOR J = 1 TO 11:Z(J) = 0: NEXT J
1520  RETURN
```

```
1530  HOME : FOR J = 1 TO 8: PRINT : NEXT J
1540  RETURN

]RUN
```

[Clear screen]

```
A SIMULATED DAILY CALORIC INTAKE AND

  ITS EFFECT ON YOUR IDEAL WEIGHT
```

[Clear screen]

```
YOU WILL BE PRESENTED A MENU FOR BREAK-

FAST, LUNCH, AND DINNER. SELECT AS MANY

ITEMS FROM EACH MENU AS YOU WISH. AFTER

YOUR DAILY MENU HAS BEEN COMPLETED, YOU

WILL RECEIVE A SUMMARY OF YOUR CALORIC

INTAKE AND ITS EFFECT ON YOUR IDEAL WT.

    D E P R E S S   A N Y   K E Y...
```

[Clear screen]

```
             BREAKFAST
  FOOD                       CALORIES

1. BACON OR SAUSAGE
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)
6. MILK
7. ORANGE JUICE
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?1
```

[Clear screen]

```
             BREAKFAST
  FOOD                       CALORIES

1. BACON OR SAUSAGE            200
2. CEREAL WITH MILK
```

```
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)
6. MILK
7. ORANGE JUICE
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?5
```

[Clear screen]

```
              BREAKFAST
  FOOD                          CALORIES

1. BACON OR SAUSAGE               200
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)                       100
6. MILK
7. ORANGE JUICE
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?6
```

[Clear screen]

```
              BREAKFAST
  FOOD                          CALORIES

1. BACON OR SAUSAGE               200
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)                       100
6. MILK                           125
7. ORANGE JUICE
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?7
```

[Clear screen]

```
              BREAKFAST
  FOOD                        CALORIES

1. BACON OR SAUSAGE             200
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)                     100
6. MILK                         125
7. ORANGE JUICE                 60
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?10
```

[Clear screen]

```
              BREAKFAST
  FOOD                        CALORIES

1. BACON OR SAUSAGE             200
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)                     100
6. MILK                         125
7. ORANGE JUICE                 60
8. PANCAKES
9. SWEET ROLL
10. TOAST                       75
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?12
```

[Clear screen]

```
                LUNCH
  FOOD                        CALORIES

1. BEER
2. BEFORE LUNCH DRINK
3. CHEESEBURGER
4. COLA
5. COTTAGE CHEESE
```

```
6. CRACKERS
7. FRENCH FRIES
8. HAMBURGER
9. MILK
10. TUNA FISH
11. VEGETABLE OR FRUIT SALAD
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?4
```

[Clear screen]

```
              LUNCH

  FOOD                        CALORIES

1. BEER
2. BEFORE LUNCH DRINK
3. CHEESEBURGER
4. COLA                         144
5. COTTAGE CHEESE
6. CRACKERS
7. FRENCH FRIES
8. HAMBURGER
9. MILK
10. TUNA FISH
11. VEGETABLE OR FRUIT SALAD
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?10
```

[Clear screen]

```
              LUNCH

  FOOD                        CALORIES

1. BEER
2. BEFORE LUNCH DRINK
3. CHEESEBURGER
4. COLA                         144
5. COTTAGE CHEESE
6. CRACKERS
7. FRENCH FRIES
8. HAMBURGER
9. MILK
10. TUNA FISH                    50
11. VEGETABLE OR FRUIT SALAD
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?12
```

[Clear screen]

```
              DINNER

  FOOD                        CALORIES

1. APPLE (OF COURSE) PIE
2. BAKED POTATO
3. BEFORE DINNER DRINK
4. BEEF STEAK
5. BEETS
6. DOZEN RAW OYSTERS
7. FISH
8. MACARONI
9. PEAS
10. TOSSED SALAD
11. T.V. DINNER
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?4
```

[Clear screen]

```
              DINNER

  FOOD                        CALORIES

1. APPLE (OF COURSE) PIE
2. BAKED POTATO
3. BEFORE DINNER DRINK
4. BEEF STEAK                   560
5. BEETS
6. DOZEN RAW OYSTERS
7. FISH
8. MACARONI
9. PEAS
10. TOSSED SALAD
11. T.V. DINNER
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?2
```

[Clear screen]

```
              DINNER

  FOOD                        CALORIES

1. APPLE (OF COURSE) PIE
2. BAKED POTATO                 250
```

```
3. BEFORE DINNER DRINK
4. BEEF STEAK                    560
5. BEETS
6. DOZEN RAW OYSTERS
7. FISH
8. MACARONI
9. PEAS
10. TOSSED SALAD
11. T.V. DINNER
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?10
```

[Clear screen]

```
              DINNER

  FOOD                           CALORIES

1. APPLE (OF COURSE) PIE
2. BAKED POTATO                  250
3. BEFORE DINNER DRINK
4. BEEF STEAK                    560
5. BEETS
6. DOZEN RAW OYSTERS
7. FISH
8. MACARONI
9. PEAS
10. TOSSED SALAD                 75
11. T.V. DINNER
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)?1
```

[Clear screen]

```
              DINNER

  FOOD                           CALORIES

1. APPLE (OF COURSE) PIE         300
2. BAKED POTATO                  250
3. BEFORE DINNER DRINK
4. BEEF STEAK                    560
5. BEETS
6. DOZEN RAW OYSTERS
7. FISH
8. MACARONI
9. PEAS
10. TOSSED SALAD                 75
```

```
11. T.V. DINNER
12. CONTINUE TO NEXT SECTION...

YOUR CHOICE(S) (1 TO 12)? 12
```

[Clear screen]

```
NOW, SOME PERSONAL DATA IS NEEDED...
```

[Clear screen]

```
ARE YOU:
   1. FEMALE
   2. MALE
ENTER 1 OR 2?2
```

[Clear screen]

```
WHAT IS YOUR HEIGHT IN INCHES?70
```

[Clear screen]

```
DO YOU CONSIDER YOURSELF:
   1. SEDENTARY (LITTLE EXERCISE)
   2. MODERATELY ACTIVE
   3. VERY ACTIVE
ENTER 1, 2, OR 3?2
```

[Clear screen]

```
       SUMMARY OF DATA

YOUR IDEAL WEIGHT IS 166

TO MAINTAIN THAT WEIGHT YOU NEED
2490 CALORIES PER DAY.

YOUR DAILY CALORIC INTAKE BASED UPON
THE LIMITED MENU IS 1935 CALORIES.

   D E P R E S S   A N Y   K E Y...
```

[Clear screen]

```
     D A T A   A N A L Y S I S

IF YOU ARE CONSISTENT IN THIS CALORIC
```

```
INTAKE YOUR WEIGHT DIFFERENTIAL WILL

BE APPROXIMATELY -1 POUNDS/WEEK.

   D E P R E S S   A N Y   K E Y...
```

[Clear screen]

```
DO YOU WISH ANOTHER ANALYSIS (Y OR N)?N
```

[Clear screen]

```
MAY YOUR BODY BE BEAUTIFUL...
```

[Clear screen]

6.6.2 PROGRAM 21: Dealing a Bridge Hand

As might be expected, one easy model to simulate is a deck of 52 cards. Our manipulation of the model will be limited to shuffling the deck and dealing one hand of 13 cards. A simulated deck may be considered as a two-dimensional array of 13 rows (card values) by 4 columns (suits). Two random-number generators can pick (deal) a given row and column, respectively, defining a position in the array. Since the random row also defines the card value (ace, deuce, etc.) and the random column defines the suit (clubs, diamonds, etc.), it is simple to PRINT the card "dealt." The position in the array may be flagged so that any dealt card will not be redealt until the deck has been "shuffled" (by reinitializing the array).

PROGRAM 21 simulates dealing a bridge hand (13 cards), and then arranging the hand by suit. This "arranging the hand by suit" introduces one example of a common programming strategy: *sorting* (see statements 880–1010). That is, let the program order a given list in either increasing or decreasing value. In the example here, the list is sorted in decreasing order by suit value (4–1) in the one-dimensional array S1(n). Thus, all the 4's are placed together, followed by all the 3's, and so on. Each suit dealt also has a card value, C1(n), where n = 1–13, assigned with it. This value is "carried" with each suit as it is sorted. (Also note another use of the IF-THEN statement in statements (650–730.)

How could the sorting routine be modified so that it would alphabetize a list of names input into one array, L$(n), as:

LAST NAME(space)FIRST NAME

and then print out the sorted (alphabetized) list? (*Note:* Program A662 on the test diskette gives one possible solution.)

RUN from disk and refer to the listing and run of PROGRAM 21.

```
]LOAD PROGRAM 21
]LIST

10  REM   PROGRAM 21
20  REM  ===============
30  REM  SIMULATION: THIS
40  REM  PROGRAM DEMOS THE USE OF A 2-DIM ARRAY
50  REM  (13 ROWS BY 4 COLUMNS) TO SIMULATE
60  REM  A CARD DECK (13 CARD VALUES BY 4 SUITS).
70  REM  THIRTEEN CARDS (E.G., A BRIDGE HAND)
80  REM  ARE RANDOMLY SELECTED FROM THE "DECK."
90  REM  A NEW USE OF THE IF-THEN STMT IS INTRODUCED
100  REM  IN COUNTING "HONOR" POINTS. THE "GET"
110  REM  STMT IS SHOWN AS AN ALTERNATIVE TO INPUT.
120  REM  THE FIRST EXAMPLE OF A SORT IS ALSO GIVEN.
130  REM  ===============
140  REM  VARIABLE DICTIONARY
150  REM  ===============
160  REM  C - CARD "VALUE" (1-13)
170  REM  C1( ) - DEALT CARD "VALUE" STORED
180  REM           FOR LATER SORTING
190  REM  C$( ) - CARD "NAME" (ACE, DEUCE, ETC.)
200  REM  C( , ) - CARD "DECK" (2-DIM ARRAY)
210  REM  P - HONOR POINT COUNTER (ACE = 4,
220  REM      KING = 3, QUEEN = 2, JACK = 1)
230  REM  S - SUIT "VALUE" (1-4)
240  REM  S1( ) - DEALT CARD SUIT "VALUE" STORED
250  REM           FOR LATER SORTING
260  REM  S$( ) - SUIT "NAME" (CLUBS, DIAMONDS, ETC.)
270  REM  ===============
280  DIM C(13,4),C$(13),S$(4),S1(14),C1(13)
290  REM  ===CARD NAMES===
300  DATA  "ACE","DEUCE","TREY","FOUR","FIVE","SIX"
310  DATA  "SEVEN","EIGHT","NINE","TEN"
320  DATA  "JACK","QUEEN","KING"
330  REM  ===SUIT NAMES===
340  DATA  "CLUBS","DIAMONDS","HEARTS","SPADES"
350  REM  ===STORE THE CARD NAMES===
360  FOR I = 1 TO 13
370  READ C$(I)
380  NEXT I
390  REM  ===STORE THE SUIT NAMES===
400  FOR I = 1 TO 4
410  READ S$(I)
420  NEXT I
430  HOME
440  PRINT "   A SIMULATED BRIDGE HAND"
450  PRINT
460  REM  ===============
470  REM  INITIALIZE THE ARRAY (SHUFFLE
```

```
480  REM  THE CARD DECK)
490  REM  ===============
500  FOR I = 1 TO 13
510  FOR J = 1 TO 4
520 C(I,J) = 0
530  NEXT J
540  NEXT I
550  PRINT "HERE'S HOW THEY WERE DEALT:"
560  PRINT
570  FOR D = 1 TO 13
580  REM  ===PICK A CARD NAME===
590 C =  INT (13 * RND (1) + 1)
600  REM  ===PICK A SUIT NAME===
610 S =  INT (4 * RND (1) + 1)
620  REM  ===HAS THIS CARD BEEN DEALT?===
630  IF C(C,S) = 1 THEN 590
640 C(C,S) = 1
650  REM  ===============
660  REM  HERE IS A NEW USE OF IF-THEN STMTS:
670  REM  IF THE EXPRESSION IS TRUE, THEN THE
680  REM  VALUE OF P WILL BE INCREASED ACCORDINGLY.
690  REM  ===============
700  IF C = 1 THEN P = P + 4: GOTO 750
710  IF C = 13 THEN P = P + 3: GOTO 750
720  IF C = 12 THEN P = P + 2: GOTO 750
730  IF C = 11 THEN P = P + 1
740  REM  ===STORE THE CARD NAME AND SUIT VALUES TO SORT===
750 S1(D) = S
760 C1(D) = C
770  PRINT  TAB( 5);C$(C); TAB( 12);"OF"; TAB( 16);S$(S)
780  NEXT D
790  PRINT
800  PRINT "DEPRESS ANY KEY, AND I'LL"
810  PRINT "ARRANGE THE HAND BY SUIT."
820  REM  ===============
830  REM  THE "GET" STATEMENT STORES THE VALUE
840  REM  OF ANY DEPRESSED KEY IN THE VARIABLE.
850  REM  ===============
860  GET Z$
870  HOME
880  REM  =========SORTING ROUTINE=========
890  FOR J = 2 TO 13
900 D1 = S1(J)
910 D2 = C1(J)
920  FOR K = J - 1 TO 1 STEP  - 1
930  IF S1(K) > = D1 THEN 980
940 S1(K + 1) = S1(K)
950 C1(K + 1) = C1(K)
960  NEXT K
970 K = 0
```

```
980 S1(K + 1) = D1
990 C1(K + 1) = D2
1000  NEXT J
1010  REM  =========END OF SORTING==========
1020  PRINT  TAB( 5);"ARRANGED BY SUIT:"
1030  PRINT
1040  FOR I = 1 TO 13
1050  PRINT TAB( 5);C$(C1(I)); TAB( 12);"OF"; TAB( 16);
     S$(S1(I))
1060  IF S1(I) = S1(I + 1) THEN 1090
1070  REM  ===SKIP A LINE BETWEEN SORTED "SUITS"===
1080  PRINT
1090  NEXT I
1100  PRINT : PRINT
1110  PRINT "HOW MANY *H O N O R* POINTS "
1120  PRINT "ARE IN THIS HAND";
1130  INPUT H
1140  IF H = P THEN 1170
1150  PRINT "I COUNT ";P;" HONOR POINTS!"
1160  GOTO 1180
1170  PRINT "GO GET 'EM, GOREN! THAT'S RIGHT!"
1180  PRINT : PRINT "DEAL ANOTHER (Y OR N)";
1190  INPUT Z$
1200  IF Z$ <  > "Y" THEN 1220
1210 P = 0: HOME : GOTO 500
1220  HOME : FOR I = 1 TO 11: PRINT : NEXT I
1230  PRINT "MAY LIFE BE A GRAND SLAM ALWAYS FOR YOU!"
1240  END

]RUN
```

[Clear screen]

```
   A SIMULATED BRIDGE HAND

HERE'S HOW THEY WERE DEALT:

     SEVEN  OF  DIAMONDS
     QUEEN  OF  CLUBS
     FIVE   OF  HEARTS
     FIVE   OF  DIAMONDS
     JACK   OF  HEARTS
     ACE    OF  CLUBS
     TREY   OF  HEARTS
     TREY   OF  CLUBS
     SIX    OF  SPADES
     SIX    OF  HEARTS
     EIGHT  OF  HEARTS
     KING   OF  CLUBS
     TEN    OF  DIAMONDS
```

```
DEPRESS ANY KEY, AND I'LL
ARRANGE THE HAND BY SUIT.
```

[Clear screen]

```
    ARRANGED BY SUIT:

    SIX    OF  SPADES

    FIVE   OF  HEARTS
    JACK   OF  HEARTS
    TREY   OF  HEARTS
    SIX    OF  HEARTS
    EIGHT  OF  HEARTS

    SEVEN  OF  DIAMONDS
    FIVE   OF  DIAMONDS
    TEN    OF  DIAMONDS

    QUEEN  OF  CLUBS
    ACE    OF  CLUBS
    TREY   OF  CLUBS
    KING   OF  CLUBS

HOW MANY *H O N O R* POINTS
ARE IN THIS HAND?9
I COUNT 10 HONOR POINTS!

DEAL ANOTHER (Y OR N)?Y
```

[Clear screen]

```
HERE'S HOW THEY WERE DEALT:

    FIVE   OF  DIAMONDS
    FOUR   OF  HEARTS
    SEVEN  OF  CLUBS
    TREY   OF  SPADES
    KING   OF  DIAMONDS
    NINE   OF  DIAMONDS
    DEUCE  OF  HEARTS
    ACE    OF  CLUBS
    FIVE   OF  CLUBS
    FOUR   OF  CLUBS
    EIGHT  OF  CLUBS
    TEN    OF  SPADES
    FOUR   OF  SPADES

DEPRESS ANY KEY, AND I'LL
ARRANGE THE HAND BY SUIT.
```

[Clear screen]

```
    ARRANGED BY SUIT:

    TREY   OF  SPADES
    TEN    OF  SPADES
    FOUR   OF  SPADES

    FOUR   OF  HEARTS
    DEUCE  OF  HEARTS

    FIVE   OF  DIAMONDS
    KING   OF  DIAMONDS
    NINE   OF  DIAMONDS

    SEVEN  OF  CLUBS
    ACE    OF  CLUBS
    FIVE   OF  CLUBS
    FOUR   OF  CLUBS
    EIGHT  OF  CLUBS

HOW MANY *H O N O R* POINTS
ARE IN THIS HAND?7
GO GET 'EM, GOREN! THAT'S RIGHT!

DEAL ANOTHER (Y OR N)?N
```

[Clear screen]

```
MAY LIFE BE A GRAND SLAM ALWAYS FOR YOU!
```

6.6.3 PROGRAM 22: Rolling a Pair of Dice

Another easy model to simulate is rolling a pair of dice. Any die rolled will give a random number, 1 through 6. The sum of the two dice is the roll value. Given an infinite number of rolls, what number is cast most often? PROGRAM 22 can provide a simulated, but nonetheless, accurate answer to this question (within roll limits). A rough plot of the percentage distribution is also shown by printing the number of asterisks on a line that corresponds to the integer value of the percentage distribution (see statements 660–700).

An interesting effect can be seen by changing statements 690 and 710 to:

```
690 INVERSE:PRINT " ";
710 NORMAL:PRINT
```

Again, the most important point in designing and developing any simulation is in defining the model. Once this is done, it may be possible to design a simulation of the model. (*Note:* For some fun and games, but a loosely based simulation, RUN ISLAND from the diskette.)

RUN from disk and refer to the listing and run of PROGRAM 22.

```
]LOAD PROGRAM 22
]LIST

10  REM   PROGRAM 22
20  REM  ==============
30  REM  SIMULATION: THIS
40  REM  PROGRAM SIMULATES ROLLING A PAIR OF DICE
50  REM  UP TO 1000 TIMES, GIVING THE DISTRIBUTION
60  REM  FOR EACH SET OF ROLLS.  EXAMINATION OF
70  REM  THE PERCENTAGE DISTRIBUTION OR ITS PLOT
80  REM  MAY BE USED TO ILLUSTRATE THE NORMAL
90  REM  DISTRIBUTION CURVE OF THE RANDOM-NUMBER
100  REM  GENERATOR, GIVEN SUFFICIENT ROLLS.
110  REM  ===============
120  REM  VARIABLE DICTIONARY
130  REM  ===============
140  REM  D1 - FIRST DIE (OF A PAIR OF DICE)
150  REM  D2 - SECOND DIE
160  REM  P( ) - COUNT OF A GIVEN VALUE FOR
170  REM         A ROLL OF A PAIR OF DICE
180  REM  P1( ) - PERCENTAGE DISTRIBUTION
190  REM  R - NUMBER OF ROLLS (VIA INPUT)
200  REM  S - SUM OF D1 AND D2 (VALUE OF A GIVEN ROLL)
210  REM  ===============
220  DIM P(12),P1(12)
230  HOME
240  PRINT "THIS PROGRAM SIMULATES"
250  PRINT "ROLLING A PAIR OF DICE"
260  REM  ===INITIALIZE THE COUNT ARRAY===
270  FOR I = 2 TO 12
280 P(I) = 0
290  NEXT I
300  PRINT
310  PRINT "HOW MANY ROLLS";
320  INPUT R
330  IF R < 1001 THEN 390
340  PRINT "IT TAKES A WHILE TO DO MORE THAN"
350  PRINT "1000 ROLLS...SO WHY DON'T YOU"
360  PRINT "GIVE ME A LOWER NUMBER..."
370  GOTO 300
380  REM  ===SHOW THE USER WE'RE DOING IT===
390  HOME : FOR I = 1 TO 10: PRINT : NEXT I: PRINT  TAB( 8)
     ;"ROLLING..."
400  REM  ===DO THE ROLLS===
410  FOR T = 1 TO R
420  REM  ===GET A RANDOM VALUE FOR EACH DIE===
430 D1 =  INT (6 *  RND (1) + 1)
440 D2 =  INT (6 *  RND (1) + 1)
```

```
450  REM  ===SUM THE PAIR OF DICE===
460 S = D1 + D2
470  REM  ===INCREASE THAT COUNT BY ONE===
480 P(S) = P(S) + 1
490  NEXT T
500  HOME
510  PRINT "VALUE OF ROLL"; TAB( 18);"COUNT"; TAB( 30);"%"
520  PRINT
530  REM  ===NOW SHOW THE DISTRIBUTION===
540  FOR L = 2 TO 12
550  REM  ===ROUND OFF THE PERCENTAGE VALUES===
560 P1(L) =  INT ((P(L) + .5) * 100 / R)
570  PRINT  TAB( 6);L; TAB( 20);P(L); TAB( 29); INT ((P(L) *
     100 / R) * 100) / 100
580  NEXT L
590  PRINT
600  PRINT "WANT TO SEE THE DISTRIBUTION"
610  PRINT "CURVE (Y OR N)";
620  INPUT A$
630  IF A$ <  > "Y" THEN 730
640  HOME : PRINT : PRINT "     P E R C E N T A G E"
650  PRINT "   D I S T R I B U T I O N": PRINT "   --------
     ----------------"
660  FOR I = 2 TO 12
670  PRINT I; TAB( 4);"I";
680  FOR J = 1 TO P1(I)
690  PRINT "*";
700  NEXT J
710  PRINT
720  NEXT I
730  PRINT : PRINT "WANT ANOTHER SET OF ROLLS (Y OR N)";
740  INPUT A$
750  IF A$ = "Y" THEN 270
760  END

]RUN
```

[Clear screen]

```
THIS PROGRAM SIMULATES
ROLLING A PAIR OF DICE

HOW MANY ROLLS?1000000
IT TAKES A WHILE TO DO MORE THAN
1000 ROLLS...SO WHY DON'T YOU
GIVE ME A LOWER NUMBER...

HOW MANY ROLLS?50
```

[Clear screen]

```
        ROLLING...
```

[Clear screen]

```
VALUE OF ROLL     COUNT        %

      2              2         4
      3              4         8
      4              3         6
      5              6         12
      6              5         10
      7              6         12
      8              7         14
      9              10        20
      10             1         2
      11             5         10
      12             1         2

WANT TO SEE THE DISTRIBUTION
CURVE (Y OR N)?Y
```

[Clear screen]

```
     P E R C E N T A G E
   D I S T R I B U T I O N
   ------------------------
2  I*****
3  I*********
4  I*******
5  I*************
6  I***********
7  I*************
8  I***************
9  I*********************
10 I***
11 I***********
12 I***

WANT ANOTHER SET OF ROLLS (Y OR N)?Y

HOW MANY ROLLS?100
```

[Clear screen]

```
        ROLLING...
```

[Clear screen]

```
VALUE OF ROLL    COUNT        %

     2             3         3
     3             5         5
     4             11        11
     5             15        15
     6             13        13
     7             14        14
     8             13        13
     9             7         7
     10            9         9
     11            7         7
     12            3         3

WANT TO SEE THE DISTRIBUTION
CURVE (Y OR N)?Y
```

[Clear screen]

```
     P E R C E N T A G E
   D I S T R I B U T I O N
   ------------------------
2  I***
3  I*****
4  I***********
5  I***************
6  I*************
7  I**************
8  I*************
9  I*******
10 I*********
11 I*******
12 I***

WANT ANOTHER SET OF ROLLS (Y OR N)?Y

HOW MANY ROLLS?1000
```

[Clear screen]

```
       ROLLING...
```

[Clear screen]

```
VALUE OF ROLL    COUNT        %

     2             27        2.7
     3             52        5.19
```

```
     4              89        8.89
     5              100       10
     6              139       13.89
     7              165       16.5
     8              144       14.39
     9              108       10.8
     10             93        9.3
     11             55        5.5
     12             28        2.79

WANT TO SEE THE DISTRIBUTION
CURVE (Y OR N)?Y
```

[Clear screen]

```
     P E R C E N T A G E
   D I S T R I B U T I O N
   ------------------------
2  I**
3  I*****
4  I********
5  I**********
6  I************
7  I****************
8  I*************
9  I**********
10 I*********
11 I*****
12 I**

WANT ANOTHER SET OF ROLLS (Y OR N)?N
```

6.7 TESTING

Testing is another application similar to drill and practice, with the exception that no "assistance" is provided. A question is asked, user response is entered, and, at some point, the user's performance is indicated.

6.7.1 PROGRAM 23: Name the Seven Dwarfs

PROGRAM 23 is a short example of a testing program. This particular program tests the naming of the seven dwarfs of Snow White fame. Names are READ into a one-dimensional array, and then a question loop asks for one of those names. An internal loop searches the list of names for a match. If a match occurs, it is checked for being previously named (flagged). At the conclusion of the program, the complete list is shown and any names in the list not given by the user are starred (*****).

Note the use of the one-dimensional array D(n), where n = 1–7, as a flag that prevents double credit for the same name being entered twice. The same flag is also used to "star" those names not entered when the test was taken (see statements 600–670 and 900–940).

Although this program tests on naming dwarfs, the program itself may be used as a general test program. By just changing the DIM, DATA, and introductory PRINT statements accordingly, the program could test naming from any chosen list.

RUN from disk and refer to the listing and run of PROGRAM 23.

```
]LOAD PROGRAM 23
]LIST

10  REM   PROGRAM 23
20  REM  ==============
30  REM  TESTING: THIS
40  REM  PROGRAM DEMOS SIMPLE TESTING EXERCISE
50  REM  IN NAMING.  PROGRAM CHECKS ANY NAME INPUT
60  REM  FIRST FOR ACCURACY AND, IF OK, THEN TO
70  REM  SEE IF NAME HAS BEEN INPUT PREVIOUSLY.
80  REM  ANY NAME NOT ANSWERED IS LISTED AT THE
90  REM  CONCLUSION OF THE PROGRAM.  BY CHANGING
100  REM  THE DIM, DATA, AND ALL "FOR-TO"
110  REM  STATEMENTS TO THE NUMBER OF NAMES IN
120  REM  THE LIST, THE PROGRAM MAY BE USED AS
130  REM  A MODEL FOR TESTING ANY LIST OF NAMES.
140  REM  ================
150  REM  VARIABLE DICTIONARY
160  REM  ================
170  REM  D( ) - FLAG FOR THE NUMBER OF THE NAME
180  REM         CORRECTLY ENTERED
190  REM  D$( ) - LIST OF NAMES
200  REM  F - FLAG FOR MISSING AT LEAST ONE NAME
210  REM  L - LENGTH OF THE LIST (NO. OF ELEMENTS)
220  REM  R - RANDOM NUMBER (4-1)
230  REM  R$(R) - POSITIVE FEEDBACK
240  REM  S - NUMBER CORRECT COUNTER (SCORE)
250  REM  ================
260  REM  ===POSITIVE FEEDBACK CHOICES===
270  DATA  "O. K.","G R E A T","S U P E R","V E R Y
     G O O D"
280  REM  ===NO. OF ITEMS IN LIST===
290  DATA  7
300  REM  ===LIST OF ITEMS TO BE NAMED===
310  DATA  "BASHFUL","DOC","DOPEY","GRUMPY"
320  DATA  "HAPPY","SLEEPY","SNEEZY"
330  REM  ===STORE THE FEEDBACK===
340  FOR I = 1 TO 4
```

```
350  READ R$(I)
360  NEXT I
370  REM  ===STORE HOW LONG THE LIST IS===
380  READ L
390  REM  ===STORE THE LIST OF NAMES===
400  FOR I = 1 TO L
410  READ D$(I)
420 D(I) = 0
430  NEXT I
440  REM  ===BEGIN THE TEST===
450  HOME
460  PRINT "SNOW WHITE AND THE 7 DWARFS"
470  PRINT
480  PRINT "LET'S SEE IF YOU CAN NAME THEM..."
490  FOR I = 1 TO 2000: NEXT I
500  HOME
510  FOR T = 1 TO L
520  FOR I = 1 TO 11: PRINT : NEXT I
530  PRINT "NAME NUMBER ";T;
540  INPUT R$
550  REM  ===============
560  REM  GO THRU THE LIST TO CHECK FOR A MATCH
570  REM  ===============
580  FOR K = 1 TO L
590  IF R$ <  > D$(K) THEN 750
600  REM  ===============
610  REM  NAME INPUT MATCHES ONE IN THE LIST, BUT
620  REM  HAS IT BEEN PREVIOUSLY ENTERED? IF NOT,
630  REM  SET THE D(K) = 1, INCREASE THE SCORE
640  REM  BY 1, AND GIVE A POSITIVE RESPONSE
650  REM  ===============
660  IF D(K) = 1 THEN 730
670 D(K) = 1
680 S = S + 1
690  HOME : FOR I = 1 TO 11: PRINT : NEXT I
700 R =  INT (4 *  RND (1) + 1): PRINT  TAB( 5);R$(R);"!"
710  FOR I = 1 TO 1000: NEXT I: HOME
720  GOTO 810
730  PRINT "YOU HAVE GIVEN THAT NAME BEFORE!"
740  GOTO 710
750  NEXT K
760  REM  ===============
770  REM  IF WE GOT THIS FAR, INPUT NAME DID
780  REM  NOT MATCH ANY NAME IN THE LIST
790  REM  ===============
800  PRINT "HMMM...THAT'S NOT ONE...":GOTO 710
810  NEXT T
820  PRINT : PRINT
830  PRINT "DEPRESS ANY KEY FOR THE COMPLETE LIST";
840  GET Z$
```

```
850  HOME
860  PRINT
870  PRINT  TAB( 8);"THE COMPLETE LIST:"
880  FOR I = 1 TO L
890  PRINT  TAB( 12);D$(I);
900  REM  ===THOSE CORRECTLY NAMED WERE "FLAGGED" (D(I) = 1)
     ===
910  IF D(I) = 1 THEN 940
920  PRINT " *****"
930 F = 1: GOTO 950
940  PRINT
950  NEXT I
960  PRINT
970  IF F = 1 THEN 1010
980  REM  ===IF F IS ZERO, ALL WERE NAMED===
990  PRINT  TAB( 6);"*** YOU KNEW THEM ALL! ***"
1000  GOTO 1030
1010  PRINT  TAB( 6);"(***** = NAME NOT LISTED!)": PRINT
1020  REM ===SHOW THE SCORE TO ONE DECIMAL PLACE===
1030  PRINT "THAT'S "; INT (S * 100 / L * 10) / 10;"
     PERCENT CORRECT!"
1040  PRINT
1050  PRINT "BYE-BYE FOR NOW...AND WATCH"
1060  PRINT "   OUT FOR THOSE APPLES!"
1070  END

]RUN
```

[Clear screen]

```
SNOW WHITE AND THE 7 DWARFS

LET'S SEE IF YOU CAN NAME THEM...
```

[Clear screen]

```
NAME NUMBER 1?SNEEZY
```

[Clear screen]

```
    G R E A T!
```

[Clear screen]

```
NAME NUMBER 2?DOC
```

[Clear screen]

```
    O. K.!
```

[Clear screen]

```
NAME NUMBER 3?GRUMPY
```

[Clear screen]

```
    V E R Y  G O O D!
```

[Clear screen]

```
NAME NUMBER 4?DUMBO
HMMM...THAT'S NOT ONE...
```

[Clear screen]

```
NAME NUMBER 5?DOPEY
```

[Clear screen]

```
    O. K.!
```

[Clear screen]

```
NAME NUMBER 6?GRUMPY
YOU HAVE GIVEN THAT NAME BEFORE!
```

[Clear screen]

```
NAME NUMBER 7?SLEEPY
```

[Clear screen]

```
    O. K.!
```

[Clear screen]

```
DEPRESS ANY KEY FOR THE COMPLETE LIST
```

[Clear screen]

```
        THE COMPLETE LIST:
            BASHFUL *****
            DOC
            DOPEY
            GRUMPY
            HAPPY *****
            SLEEPY
            SNEEZY

      (***** = NAME NOT LISTED!)
```

```
THAT'S 71.4 PERCENT CORRECT!

BYE-BYE FOR NOW...AND WATCH
   OUT FOR THOSE APPLES!
```

6.7.2 PROGRAM 24: Multiple-Choice Questions

PROGRAM 24 is one example of generating multiple-choice questions. Following any introductory statements, PRINT statements that ask questions and DATA statements that provide the choices and their appropriate responses may be added to the program. The correct choice by number is assigned to variable A, and then a GOSUB transfers to a subroutine that displays the choices and evaluates the user's input. This sequence of PRINT (the question), DATA (for each choice and its response), A = (number of the correct choice), and GOSUB 5000 may be repeated for an indefinite number of multiple-choice questions in the program.

(This program arbitrarily presents only 4 choices. What would be needed to change the program so that 5 choices would be displayed?)

RUN from disk and refer to the listing and run of PROGRAM 24.

```
]LOAD PROGRAM 24
]LIST

10  REM   PROGRAM 24
20  REM  ===============
30  REM  TESTING: THIS
40  REM  PROGRAM DEMOS MULTIPLE-CHOICE TESTING
50  REM  USING DATA-READ TECHNIQUES.  QUESTIONS
60  REM  ARE ASKED IN SEQUENCE (I.E., NO RANDOMIZATION).
70  REM  ALL QUESTIONS "SETS" ARE ENTERED IN THE
80  REM  PROGRAM FOLLOWING THE SEQUENCE:
90  REM       1. PRINT STATEMENTS TO ASK THE QUESTION
100  REM      2. DATA STATEMENTS FOR 4 CHOICES AND
110  REM         THE RESPONSE FOR EACH CHOICE
120  REM      3. SETTING VARIABLE "A" TO THE
130  REM         NUMBER OF THE CORRECT CHOICE
140  REM      4. GOSUB 5000
150  REM  QUESTION "SETS" ON ANY TOPIC MAY BE
160  REM  USED IN THE PROGRAM IF THIS SEQUENCE
170  REM  IS FOLLOWED.
180  REM  ===============
190  REM  VARIABLE DICTIONARY
200  REM  ===============
210  REM  A - CORRECT CHOICE ANSWER (1-4)
220  REM  A$( ) - A GIVEN CHOICE (READ FROM DATA)
230  REM  C - NUMBER CORRECT COUNTER
240  REM  R - USER'S ANSWER (VIA INPUT)
```

```
250  REM  R$( ) - A GIVEN RESPONSE (READ FROM DATA)
260  REM  ===============
270  DIM A$(4),R$(4)
280  HOME :C = 0
290  REM  ===============
300  REM  ADD INTRODUCTORY STATEMENTS, EXAMPLES,
310  REM   OR WHATEVER HERE (UP TO LINE 500)
320  REM  ===============
330  REM
340  REM
500  REM  ==============
510  REM  PRINT THE QUESTION
520  REM  ==============
530  PRINT "THE STATE FLOWER OF TEXAS IS THE:"
540  REM  ==============
550  REM  ADD 4 DATA ELEMENT PAIRS FOR EACH
560  REM  CHOICE AND THE RESPONSE FOR THAT CHOICE
570  REM  ===============
580  DATA  "BLUE-BONNET","BEAUTIFUL, AREN'T THEY"
590  DATA  "ROSE","IT'S BY ANOTHER NAME HERE"
600  DATA  "DANDELION","BLOW IT OFF"
610  DATA  "MORNING GLORY","IT AIN'T, BUT IT COULD BE"
620  REM  ===============
630  REM  SET VARIABLE "A" TO THE CORRECT CHOICE NUMBER
640  REM  ===============
650 A = 1
660  REM  ===============
670  REM  THEN GOSUB 5000 TO PRINT THE CHOICES,
680  REM  GET THE ANSWER, AND THEN RESPOND TO IT.
690  REM  ===============
700  GOSUB 5000
710  REM  ===NEXT QUESTION SEQUENCE, ETC.===
720  PRINT "AN EXAMPLE OF A COMPUTER OUTPUT"
730  PRINT "DEVICE IS:"
740  DATA  "PRINTER","YES, BUT THERE WAS ANOTHER IN THE
     LIST"
750  DATA  "KEYBOARD","THAT'S AN INPUT DEVICE!"
760  DATA  "TERMINAL SCREEN","YES, BUT THERE IS A BETTER
     CHOICE"
770  DATA  "1. AND 3. ABOVE","O.K...THEY ARE 2 COMMON
     EXAMPLES"
780 A = 4
790  GOSUB 5000
800  REM  ===NEXT QUESTION SEQUENCE, ETC.===
810  PRINT "WHICH PLANET IS EARTH"
820  PRINT "FROM THE SUN?"
830  DATA  "FIRST","THAT'S MERCURY"
840  DATA  "SECOND","'TIS VENUS, ABOUT DE MILO FROM THE
     SUN!"
850  DATA  "THIRD","RIGHT...YOU'RE A TERROR FARMER"
```

```
860  DATA  "FOURTH","MAR-CY, THAT'S MARS"
870 A = 3
880  GOSUB 5000
890  REM  ===============
900  REM  ROOM TO ADD MANY MORE QUESTION "SETS"
910  REM  FOLLOWING THE SEQUENCE OF:
920  REM  PRINT, DATA, A = , GOSUB
930  REM  ===============
4800  REM
4810  REM
4820  REM  =================
4830  REM  IT'S OK TO HAVE THE 'END' STATEMENT
4840  REM  ** NOT ** AS THE LAST STATEMENT!
4850  REM  =================
4860  PRINT : PRINT "YOU ANSWERED ";C;" CORRECTLY!"
4870  END
4880  REM  ===============
4890  REM  SUBROUTINE TO DISPLAY THE CHOICES, STORE
4900  REM  THE RESPONSE FOR EACH CHOICE, AND
4910  REM  GET THE INPUT FOR CHECKING
4920  REM  ===============
5000  PRINT
5010  REM  ==============
5020  REM  READ THE DATA FOR THE CHOICE AND
5030  REM  ITS RESPONSE; PRINT THE CHOICE
5040  REM  ==============
5050  FOR I = 1 TO 4
5060  READ A$(I),R$(I)
5070  PRINT I;". ";A$(I)
5080  PRINT
5090  NEXT I
5100  PRINT
5110  PRINT "YOUR CHOICE (1-4)";
5120  INPUT R
5130  REM  ===CHECK FOR WITHIN RANGE===
5140  IF R < 1 THEN 5110
5150  IF R > 4 THEN 5110
5160  PRINT
5170  REM  ===PRINT THE RESPONSE FOR USER'S CHOICE===
5180  PRINT R$(R);"!"
5190  REM  ===IS IT THE CORRECT CHOICE?===
5200  IF A = R THEN 5300
5210  REM  ==============
5220  REM  IF THE USER'S CHOICE IS NOT CORRECT,
5230  REM  PRINT THE CORRECT CHOICE NUMBER
5240  REM  AND THE CHOICE LISTED
5250  REM  ===============
5260  PRINT
5270  PRINT "A CORRECT CHOICE IS ";A;".: ";A$(A)
5280  GOTO 5320
```

```
5290  REM ===INCREASE A NUMBER-CORRECT COUNTER===
5300 C = C + 1
5310  REM  ===LET THE USER SAY WHEN TO GO ON===
5320  PRINT
5330  PRINT "DEPRESS ANY KEY TO CONTINUE..."
5340  GET Z$
5350  HOME
5360  REM  ===RETURN FOR THE NEXT QUESTION===
5370  RETURN

]RUN PROGRAM 24
```

[Clear screen]

```
THE STATE FLOWER OF TEXAS IS THE:

1. BLUE-BONNET

2. ROSE

3. DANDELION

4. MORNING GLORY

YOUR CHOICE (1-4)?2

IT'S BY ANOTHER NAME HERE!

A CORRECT CHOICE IS 1.: BLUE-BONNET

DEPRESS ANY KEY TO CONTINUE...
```

[Clear screen]

```
AN EXAMPLE OF A COMPUTER OUTPUT
DEVICE IS:

1. PRINTER

2. KEYBOARD

3. TERMINAL SCREEN

4. 1. AND 3. ABOVE

YOUR CHOICE (1-4)?2

THAT'S AN INPUT DEVICE!!

A CORRECT CHOICE IS 4.: 1. AND 3. ABOVE

DEPRESS ANY KEY TO CONTINUE...
```

[Clear screen]

```
WHICH PLANET IS EARTH
FROM THE SUN?

1. FIRST

2. SECOND

3. THIRD

4. FOURTH

YOUR CHOICE (1-4)?2

'TIS VENUS, ABOUT DE MILO FROM THE SUN!

A CORRECT CHOICE IS 3.: THIRD

DEPRESS ANY KEY TO CONTINUE...
```

[Clear screen]

```
YOU ANSWERED 0 CORRECTLY!
```

6.8 THE KEYWORD SUBROUTINE

Up to this point, a variety of instructional computing program examples and models have been presented. These programs illustrate some of the major concepts, strategies, and techniques that may be used in program design. However, one additional technique that merits discussion is *keyword matching*.

This technique allows a program author to define a "keyword" sequence of characters that, if found anywhere in the user's response in the same sequence, will constitute a match between the input and an anticipated answer. For example, assume that an author wanted to ask the following question:

```
YOU HAVE REMOVED YOUR DIRTY SOCKS.
WHAT SHOULD YOU DO WITH THEM NOW?
```

Further assume that the author anticipates the following responses as possible answers to the question:

```
WASH AND DRY THEM
WASH THEM
THROW THEM AWAY
GIVE THEM AWAY
```

With the use of a keyword subroutine, the author can define a match of these anticipated answers as:

```
"WASH*DRY"
"WASH"
"THROW"
"GIVE"
```

Thus, if the user responds with *any* phrase containing the word GIVE, for example, then a match will have been found. The author can then have the program make appropriate responses and transfer execution back to the original question or give the complete answer. If none of the anticipated answers are matched, a response (such as a hint) can be made and execution transferred accordingly.

The program fragment, KEYWORD, found on the text diskette is one example of a subroutine of this nature. Although many of the program statements are beyond the scope of this text, it is very easy to use the subroutine. However, since KEYWORD is already written, there are certain conventions that *must* be followed for its successful use:

1. The user's response must be in R$ (i.e., INPUT R$).
2. The defined anticipated answers (keywords to search for in the user's response) must be assigned to A$.
3. A$ may have as many as three keywords, delimited (separated) by an asterisk.
4. If a match occurs between the anticipated answer and the user's response, A$ is set to "0" (string zero). If no match is found, A$ is set to "1" (string one). Appropriate branching in the program is then based upon the value of A$.
5. The subroutine begins with statement number 5000.
6. The END statement is number 10000.

Refer to the creation and run of the program socks.

```
]LOAD KEYWORD

]1 REM  ===PROGRAM NAME: SOCKS===

]2 REM  ===DEMOS KEYWORD SUBROUTINE===

]3 REM  ===VARIABLE F IS A COUNTER FOR===

]4 REM  ===REPEATING THE QUESTION NO===

]5 REM  ===MORE THAN 4 TIMES===
```

KEYWORD is loaded from diskette [i.e., KEYWORD subroutine (statements in range 5000 through 10000) is loaded into system's memory].

1–5 added to document program.

```
]6 F = 0

]10 F = F + 1

]20 IF F > 4 THEN 230

]30 PRINT

]40 PRINT "YOU HAVE REMOVED YOUR DIRTY"

]50 PRINT "SOCKS, WHAT SHOULD YOU DO"

]60 PRINT "WITH THEM NOW";

]70 INPUT R$

]80 A$ = "WASH*DRY":GOSUB 5000

]90 IF A$ = "1" THEN 110

]100 PRINT "GOOD!  YOU MIGHT USE A BIT"

]105 PRINT "OF FOOT POWDER, TOO!":GOTO 10000

]110 A$ = "WASH":GOSUB 5000

]120 IF A$ = "1" THEN 140

]130 PRINT "DO YOU WEAR WET SOCKS?":GOTO 10

]140 A$ = "THROW":GOSUB 5000

]150 IF A$ = "1" THEN 180

]160 PRINT "DON'T TOSS THEM YET...TRY"

]170 PRINT "SOME SOAP AND WATER.":GOTO 10

]180 A$ = "GIVE":GOSUB 5000

]190 IF A$ = "1" THEN 210

]200 PRINT "NO ONE WOULD TAKE THEM!!!":GOTO 10

]210 PRINT "(NO MATCH YET...)"

]220 PRINT "THINK OF SOAP AND SUNSHINE!":GOTO 10

]230 PRINT:PRINT

]240 PRINT "THOSE DIRTY SOCKS SHOULD BE"
```

20 limits number of times question asked to 4.

30–60 added to ask the question.

70 assigns user's response to R$.

80 assigns first anticipated answer (in this case, correct response) to A$. Two keywords needed, WASH and DRY, delimited by an asterisk. Transfer to subroutine at 5000.

Upon return from subroutine, 90 checks value of A$. If user's response contained *at least* keyword WASH followed somewhere by keyword DRY, a match occurred and A$ was set to "0" by subroutine. If no match, A$ was set to "1".

100–105 executed if A$ = "0" (i.e., match found). Transfer is then to 10000, the END of program.

110 executed if no match for first keyword (A$ = "1"). A$ redefined as next keyword to check for in user's response. Transfer back to subroutine. This sequence—define keyword, go to subroutine, check A$ upon return, and branching or responding accordingly—is repeated through 200.

210–220 executed if no defined keywords matched. Transfer then made back to 10 to either ask question again or give correct answer, based upon value of F.

```
]250 PRINT "WASHED AND DRIED!!!!":GOTO 10000

]RUN

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?THROW THEM IN THE LAUNDRY
DON'T TOSS THEM YET...TRY
SOME SOAP AND WATER.

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?WELL...WASH THEM I GUESS
DO YOU WEAR WET SOCKS?

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?OK..GIVE THEM TO ANYBODY
NO ONE WOULD TAKE THEM!!!

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?LET MOM WORRY ABOUT THEM
(NO MATCH YET...)
THINK OF SOAP AND SUNSHINE!

THOSE DIRTY SOCKS SHOULD BE
WASHED AND DRIED!!!!

]RUN

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?HAND THEM TO MOM
(NO MATCH YET...)
THINK OF SOAP AND SUNSHINE!

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?OH...I'D BETTER WASH AND DRY
THEM!!!
GOOD!  YOU MIGHT USE A BIT
OF FOOT POWDER, TOO!

]SAVE SOCKS
```

Following this same sequential strategy, a variety of both anticipated correct and incorrect answers may be used in a program. Program KEYWORD DEMO on the text diskette is another example of using the keyword subroutine. A sample run is shown, followed by a listing of the subroutine.

Refer to the run of the program keyword demo.

```
]RUN KEYWORD DEMO
A DEMO OF THE KEYWORD SUBROUTINE

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?UNITED
UNITED WHAT OF WHAT???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?UNITED STATES
UNITED STATES OF WHAT???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?AMERICA
YES, BUT WHAT OF AMERICA???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?U S A
USA, YES...BUT SPELL IT OUT PLEASE!

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?THE HOME OF THE FREE

THE ANSWER I WANTED WAS THE
UNITED STATES OF AMERICA!

]RUN
A DEMO OF THE KEYWORD SUBROUTINE.

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?NEVER THE UNITED STATES!!!
YOU ARE TRYING TO BE TRICKY!

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?MUST BE THE UNITED STATES OF AMERICA
THAT'S IT...VERY GOOD!

]LOAD KEYWORD
]LIST

5000  REM  ====================
5002  REM  PROGRAM NAME: KEYWORD
5004  REM  ====================
5006  REM  THIS SUBROUTINE READS A
5008  REM  USER'S RESPONSE (MUST BE
5010  REM  FROM: INPUT R$) AND CHECKS
5012  REM  FOR A "KEYWORD" CHARACTER
5014  REM  SEQUENCE MATCH AS DEFINED
5016  REM  IN A$.  IF A MATCH OCCURS.
```

```
5018  REM  A$ IS SET TO "0"; OTHERWISE, "1".
5020  REM  NOTE: A$ MAY CONTAIN UP TO 3
5030  REM  KEYWORDS DELIMITED BY *.
5040  REM  ====================
5042  REM  THE SUBROUTINE MAY BE USED WITH ANY
5044  REM  PROGRAM BY FIRST LOADING THE
5046  REM  "KEYWORD" PROGRAM, AND THEN ADDING
5048  REM  STATEMENTS IN THE SEQUENCE:
5050  REM  PRINT(S) (FOR THE QUESTION)
5052  REM  INPUT R$ (FOR THE RESPONSE)
5054  REM  A$ = "DEFINED*KEYWORD*ANSWER"
5056  REM  GOSUB 5000
5058  REM  IF A$ = "1" THEN (TO NEXT KEYWORD)
5060  REM  PRINT(S) (TO REPLY TO MATCH JUST MADE)
5062  REM  GOTO (REPEAT OF THE QUESTION,
5064  REM       OR GIVE THE ANSWER)
5066  REM  A$ = "NEXT*KEYWORD"
5068  REM  GOSUB 5000  ETC., ETC.
5070  REM  ====================
5080  REM  THE FOLLOWING DIM
5090  REM  STATEMENT MUST BE
5100  REM  DECLARED IN THE MAIN
5110  REM    PROGRAM:  DIM W$(3)
5120  REM  ====================
5130  REM  INPUT A$ (AS DEFINED)
5140  REM  ====================
5150  REM  OUTPUT - A$ = 0 CORRECT
5160  REM           A$ = 1 INCORRECT
5170  REM  ====================
5180  REM  VARIABLE DICTIONARY
5190  REM  ====================
5200  REM  A$ - KEYWORD(S) ANTICIPATED
5210  REM  LA - LENGTH OF ANSWER
5220  REM  LR - LENGTH OF RESPONSE
5230  REM  LW - LENGTH OF KEYWORD
5240  REM  NW - NO. OF KEYWORDS
5250  REM  P1,P2 - STRING POINTERS
5260  REM  R$ - USER'S RESPONSE
5270  REM  W$(3) - ARRAY FOR KEYWORDS
5280  REM  ====================
5300 LA =  LEN (A$)
5310 LR =  LEN (R$)
5320  IF LR < LA GOTO 5620
5330  FOR I = 1 TO 3
5340  FOR J = 1 TO LA
5350 P1 = J
5360  IF  MID$ (A$,J,1) = "*" GOTO 5400
5370  NEXT J
5380 P2 = P1
5390  GOTO 5410
```

```
5400 P2 = P1 - 1
5410 W$(I) = MID$ (A$,1,P2)
5420  IF P1 <  > LA GOTO 5450
5430 NW = I
5440  GOTO 5500
5450 A$ =  MID$ (A$,P1 + 1,LA - P1)
5460 LA =  LEN (A$)
5470  NEXT I
5480  PRINT "ERROR - A$ CONTAINS MORE THAN 3 WORDS"
5490  END
5500 P1 = 1
5510  FOR I = 1 TO NW
5520 LW =  LEN (W$(I))
5530  FOR J = P1 TO LR - LW + 1
5540 P2 = J
5550  IF  MID$ (R$,J,LW) = W$(I) GOTO 5580
5560  NEXT J
5570  GOTO 5620
5580 P1 = P2 + LW + 1
5590  NEXT I
5600 A$ = "0"
5610  RETURN
5620 A$ = "1"
5630  RETURN
10000  END
```

6.9 USING BASIC COMMANDS WITHIN A PROGRAM

By now, you are well aware of many of the BASIC commands (CATALOG, LOAD, RUN, LIST, etc.) of the Apple. There is a method by which commands may be incorporated *as statements* within the body of a BASIC program. When one of these statements is executed it could, for example, display the catalog of files on the disk or execute (RUN) a given program. The second example is particularly useful in that one program can "command" another program to RUN, which could command another program to RUN, and so on. Thus, programs can be "linked" together in sort of a "chain" fashion.

A common use in instructional computing is to design one program as a "menu" of available programs on the disk. When this menu program is executed, a selection of programs is displayed, and the user may enter the choice desired. Based upon the user's input, execution is transferred to the appropriate line number in the menu program that commands the system to RUN the selected program.

Without going into detail, we shall simply state here that a variable needs to be defined as a "control-D" character. One way to accomplish this is by the use of the statement CHR$(4) (see Appendix B). Thus, the statement

```
D$ = CHR$(4)
```

assigns a value of "control-D" to the variable D$ (any legal string variable name could be used). Executing the "control-D" as a PRINT statement that includes the appropriate command enclosed in quotes allows the command to literally be a statement in the program. For example, when the statement

```
PRINT D$;"RUN NEXT"
```

is executed [and D$ has an assigned value of CHR$(4)], the system will automatically LOAD and RUN the program named NEXT.

This use of commands as BASIC statements is illustrated in program MENU on the disk. Carefully examine the listings of MENU, CHAIN 1, and CHAIN 2; then, RUN MENU and note the options and actions it provides.

```
]LOAD MENU
]LIST

10  REM    PROGRAM 'MENU'
20  REM   ================
30  REM   THIS PROGRAM DEMONSTRATES THE USE OF
40  REM   INCORPORATING BASIC COMMANDS INTO THE
50  REM   BODY OF A PROGRAM BY DEFINING A
60  REM   STRING VARIABLE (E.G., D$) = CHR$(4).
70  REM   TWO PROGRAMS, CHAIN 1 AND CHAIN 2 ARE
80  REM   USED FOR DEMONSTRATION PURPOSES.
85  REM   ================
90  HOME : FOR I = 1 TO 5: PRINT : NEXT I
100  PRINT "          M E N U": PRINT
110  PRINT "THIS PROGRAM DEMONSTRATES HOW A 'MENU'"
120  PRINT "OF PROGRAMS MAY BE PRESENTED FOR"
130  PRINT "SELECTION AND THEN AUTOMATICALLY"
140  PRINT "EXECUTED (RUN) BY THE SYSTEM."
150  REM   ===============
160  REM   DEFINE A VARIABLE AS A 'CONTROL-D' (CHR$(4))
170  REM   ===============
180  D$ =  CHR$ (4)
190  PRINT : PRINT
200  PRINT "     YOUR OPTIONS:"
210  PRINT "     ---- --------"
220  PRINT "       1. PROGRAM CHAIN 1"
230  PRINT "       2. PROGRAM CHAIN 2"
240  PRINT "       3. STOP
250  PRINT "     YOUR CHOICE (1-3)?"
260  GET Z
270  IF Z >  = 1 AND Z <  = 3 THEN 330
280  PRINT "*** OUT OF RANGE ***": GOTO 250
290  REM   ===============
300  REM   CLEAR THE SCREEN, CENTER, AND
310  REM   TELL WHAT'S HAPPENING...
320  REM   ===============
```

```
330  HOME : FOR I = 1 TO 11: PRINT : NEXT I
340  ON Z GOTO 350,370,390
350  PRINT "GOING TO PROGRAM 'CHAIN 1'..."
360  PRINT D$;"RUN CHAIN 1"
370  PRINT "GOING TO PROGRAM 'CHAIN 2'..."
380  PRINT D$;"RUN CHAIN 2"
390  PRINT "STOPPING THIS INTERACTION AND"
400  PRINT "GETTING THE DISK 'CATALOG'..."
410  FOR I = 1 TO 3000: NEXT I
420  PRINT D$;"CATALOG"
430  END

]LOAD CHAIN 1
]LIST

10  REM    PROGRAM 'CHAIN 1'
20  REM   ====================
30  HOME : FOR I = 1 TO 8: PRINT : NEXT I
40  PRINT "AND HERE WE ARE EXECUTING"
50  PRINT "PROGRAM 'CHAIN 1'..."
60  PRINT
70  PRINT "IF YOU DEPRESS THE LETTER 'N'"
80  PRINT "(FOR 'NEXT'), WE'LL GO TO THE"
90  PRINT "PROGRAM 'CHAIN 2'.  ANY OTHER KEY"
100  PRINT "WILL TAKE YOU BACK TO THE 'MENU'..."
110 D$ =  CHR$ (4)
120  GET Z$
130  HOME : FOR I = 1 TO 11: PRINT : NEXT I
140  PRINT "YOU DEPRESSED THE LETTER '";Z$;"'..."
150  PRINT : PRINT "          SO..."
160  IF Z$ = "N" THEN 180
170  PRINT D$;"RUN MENU"
180  PRINT D$;"RUN CHAIN 2"
190  END

]LOAD CHAIN 2
]LIST

10  REM    PROGRAM 'CHAIN 2'
20  REM   ====================
30  HOME : FOR I = 1 TO 10: PRINT : NEXT I
40  PRINT "WELL, WE MADE IT TO PROGRAM"
50  PRINT "'CHAIN 2'...SO YOU SEE IT'S"
60  PRINT "SIMPLE TO HAVE THE SYSTEM"
70  PRINT "FOLLOW YOUR COMMANDS **IN**"
80  PRINT "A PROGRAM (IF YOU KNOW THE RULES...)"
90  PRINT
100  PRINT "NOW DEPRESS ANY KEY, AND WE'LL"
110  PRINT "GO BACK TO THE 'MENU' PROGRAM..."
120  GET Z$
```

```
130  HOME : FOR I = 1 TO 11: PRINT : NEXT I
140  PRINT "HERE WE GO BACK TO THE MENU..."
150 D$ = CHR$ (4)
160  PRINT D$;"RUN MENU"
170  END

]RUN MENU
```

[Clear screen]

```
        M  E  N  U

THIS PROGRAM DEMONSTRATES HOW A 'MENU'
OF PROGRAMS MAY BE PRESENTED FOR
SELECTION AND THEN AUTOMATICALLY
EXECUTED (RUN) BY THE SYSTEM.

     YOUR OPTIONS:
     ---- --------
       1. PROGRAM CHAIN 1
       2. PROGRAM CHAIN 2
       3. STOP
     YOUR CHOICE (1-3)? 1
```

[Clear screen]

```
GOING TO PROGRAM 'CHAIN 1'...
```

[Clear screen]

```
AND HERE WE ARE EXECUTING
PROGRAM 'CHAIN 1'...

IF YOU DEPRESS THE LETTER 'N'
(FOR 'NEXT'), WE'LL GO TO THE
PROGRAM 'CHAIN 2'.  ANY OTHER KEY
WILL TAKE YOU BACK TO THE 'MENU'...
```

[Clear screen]

```
YOU DEPRESSED THE LETTER 'N'...

          SO...
```

[Clear screen]

```
WELL, WE MADE IT TO PROGRAM
'CHAIN 2'...SO YOU SEE IT'S
SIMPLE TO HAVE THE SYSTEM
FOLLOW YOUR COMMANDS **IN**
A PROGRAM (IF YOU KNOW THE RULES...)
```

```
NOW DEPRESS ANY KEY, AND WE'LL
GO BACK TO THE 'MENU' PROGRAM...
```

[Clear screen]

```
HERE WE GO BACK TO THE MENU...
```

[Clear screen]

```
        M  E  N  U

THIS PROGRAM DEMONSTRATES HOW A 'MENU'
OF PROGRAMS MAY BE PRESENTED FOR
SELECTION AND THEN AUTOMATICALLY
EXECUTED (RUN) BY THE SYSTEM.

     YOUR OPTIONS:
     ---- --------
       1. PROGRAM CHAIN 1
       2. PROGRAM CHAIN 2
       3. STOP
     YOUR CHOICE (1-3)? 3
```

[Clear screen]

```
STOPPING THIS INTERACTION AND
GETTING THE DISK 'CATALOG'...

DISK VOLUME 254

*A 002 HELLO
*B 034 TITLE
*B 034 CREDITS
*A 002 PROGRAM 1
*A 003 PROGRAM 2
*A 006 PROGRAM 3
*A 008 PROGRAM 4
*A 008 PROGRAM 5
*A 009 PROGRAM 6
*A 009 PROGRAM 7
*A 011 PROGRAM 8
*A 005 PROGRAM 9
*A 008 PROGRAM 10
*A 018 PROGRAM 11
*A 021 PROGRAM 12
*A 007 PROGRAM 13
*A 022 PROGRAM 14
*A 009 PROGRAM 15
*A 013 PROGRAM 16
*A 022 PROGRAM 17
*A 016 PROGRAM 18
```

```
*A 011 PROGRAM 19
*A 021 PROGRAM 20
*A 014 PROGRAM 21
*A 009 PROGRAM 22
*A 012 PROGRAM 23
*A 016 PROGRAM 24
*A 006 PROGRAM 25
*A 011 PROGRAM 26
*A 006 RECORD INITIALIZER
 T 002 TESTS
*A 008 A354
*A 004 A422
*A 006 A458
*A 007 A662
*A 004 A784
*A 004 A785
*A 004 A786
*A 009 KEYWORD
*A 016 KEYWORD DEMO
*A 012 SOCKS
*A 006 MENU
*A 003 CHAIN 1
*A 003 CHAIN 2
*A 004 START
*A 004 WARNING
*A 021 ISLAND
```

Question: How could the menu of available programs on a disk be automatically displayed when the disk is loaded and the system booted up? (One solution is shown on the text disk in statement 90 of the HELLO program and by the program START.)

6.10 POSERS AND PROBLEMS

1. Identify an area in your particular field of interest in which an instructional computing program could be written for each of the five applications described above. Briefly outline each program by describing the area, content, and application in a short paragraph.

"What is the use of a book," thought Alice, "without pictures or conversations?"
—Lewis Carroll

Think About This (for Fun)

A single English word can be formed from these letters. What is it? Use all the letters: PNLLEEEESSSSS.

Think About This (Seriously)

Is is possible that graphics do not always enhance instructional computing materials?

Chapter 7

One Picture Is Worth Ten Thousand Words

7.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Explain and give an example of how to specify a point on a graphics screen (Section 7.2).
2. Define the purpose and give at least one example of each of the low-resolution graphics statements GR, COLOR, PLOT, HLIN, VLIN, and TEXT (Section 7.3).
3. Define the purpose and give at least one example of each of the high-resolution graphics statements HGR, HCOLOR, and HPLOT (Section 7.4).

4. Design, enter, and RUN a BASIC program of your own choosing using low-resolution graphics.
5. Design, enter and RUN a BASIC program of your own choosing using high-resolution graphics.

7.2 WHAT ARE GRAPHICS?

Throughout history, progress has been the result of people's ability to understand complex concepts. Visual tools such as drawings, photographs, films, and video tapes provide the medium for making complex concepts understandable to the masses. With the development of the computer and its ability to analyze vast amounts of data rapidly, its use as a tool for portraying visual information (graphics) naturally evolved.

A computer graphic is somewhat like a printed map. Both are two-dimensional surfaces with a vertical direction and a horizontal direction. Just as any point on a map may be identified by its horizontal and vertical coordinates (latitude and longitude), any point on a computer's graphics screen can be specified by measuring its vertical and horizontal distances from the upper left corner.

The horizontal distance scale is called the x-axis and the vertical distance scale is called the y-axis. Figure 7.1 shows the Apple low-resolution graphics screen with the x-axis across the top of the screen and the y-axis down the left side of the screen. When the position of a point is specified, the distance on the x-axis is specified first, followed by the distance on the y-axis (the *x- and y-coordinates* of the point). For example, 20,10 specifies the point 20 units to the right in the x-direction and 10 units down in the y-direction. Similarly, the corners of the Apple low-resolution graphics screen are specified by 0,0 (upper left), 39,0 (upper right), 39,39 (lower right), and 0,39 (lower left).

7.3 STATEMENTS FOR LOW-RESOLUTION GRAPHICS

7.3.1 Statement GR

Purpose The GR statement is used to initialize the low-resolution graphics screen in a program. When it is executed, the computer monitor will change from text to low-resolution graphics, and the screen will be cleared to black. As pictured in Figure 7.1, the low-resolution screen contains 160 points (0 to 39 by 0 to 39). In addition, four text lines are available at the bottom of the screen for instructions, questions, and comments.

7.3.2 Statement COLOR

Purpose The COLOR statement sets the color for subsequent graphics statements. Once the color has been set, all graphics drawn on the screen will be

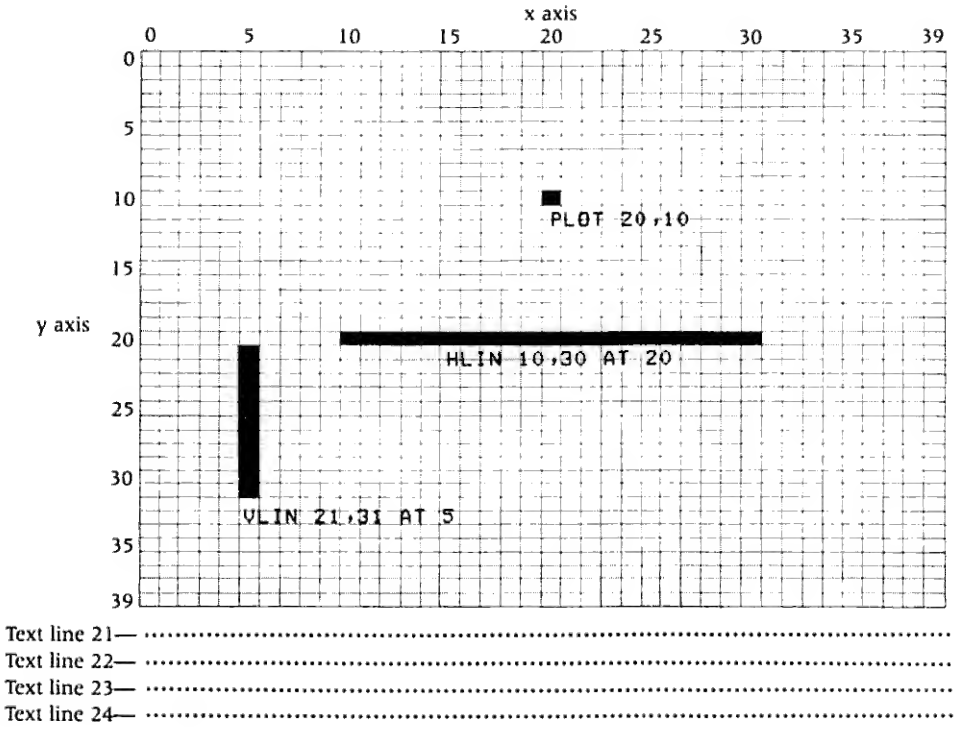


Figure 7.1
Low-resolution graphics screen (GR).

of that color until another COLOR statement is executed. Sixteen colors are available. Each color is represented by a number from 0 to 15:

`COLOR = 0`	(black)	`COLOR = 8`	(brown)
`COLOR = 1`	(magenta)	`COLOR = 9`	(orange)
`COLOR = 2`	(dark blue)	`COLOR = 10`	(grey)
`COLOR = 3`	(purple)	`COLOR = 11`	(pink)
`COLOR = 4`	(dark green)	`COLOR = 12`	(green)
`COLOR = 5`	(grey)	`COLOR = 13`	(yellow)
`COLOR = 6`	(medium blue)	`COLOR = 14`	(aqua)
`COLOR = 7`	(light blue)	`COLOR = 15`	(white)

7.3.3 Statement PLOT

Purpose The PLOT statement will place a rectangular "brick" on the screen at the x- and y-coordinates specified in the statement. The color of the brick will be the color specified by the most recently executed COLOR statement.

Example: `PLOT 20,10`

(A brick will be PLOTted at a point 20 units to the right on the x-axis and 10 units down on the y-axis. See Figure 7.1.)

Example: Enter the following program and RUN it:

```
NEW
10 GR
20 FOR I = 1 TO 15
30 COLOR = I
40 X = INT(RND(1)*40)
50 Y = INT(RND(1)*40)
60 PLOT X,Y
70 NEXT I
80 END
```

10 initializes low-resolution graphics screen.

20 defines a loop to be executed 15 times.

30 resets COLOR to new value each time through loop.

40 and 50 generate random values (0–39) for x- and y-coordinates.

60 PLOTS brick on screen.

70 and 80 continue loop and END program.

What happened? If entered correctly, fifteen bricks of different colors were PLOTted on the screen.

7.3.4 Statement HLIN

Purpose The HLIN statement draws a horizontal line on the screen from a specified starting point on the x-axis, to a specified ending point on the x-axis. The line is located at a specified y-axis point. The color of the line will be the color indicated by the most recently executed COLOR statement.

Example: `HLIN 10,30 AT 20`

(A horizontal line will be drawn from the 10th to the 30th unit on the x-axis at the 20th unit down the y-axis. See Figure 7.1.)

7.3.5 Statement VLIN

Purpose The VLIN statement draws a vertical line on the screen from a specified starting point on the y-axis to a specified ending point on the y-axis. The line is located at a specified x-axis point. The color of the line will be the color indicated by the most recently executed COLOR statement.

Example: `VLIN 21,31 AT 5`

(A vertical line will be drawn from the 21st to the 31st unit on the y-axis at the 5th unit to the right on the x-axis. See Figure 7.1.)

7.3.6 Statement TEXT

Purpose The TEXT statement returns the computer's monitor to a full text screen (24 lines of 40 characters). If this statement is not included at the end of a program using graphics, the graphics screen will remain on the monitor.

TEXT may also be typed as an individual command to return to the full text screen.

7.3.7 PROGRAM 25: Random Colored Lines

The five low-resolution graphics statements GR, COLOR, PLOT, HLIN, and VLIN provide the basis for adding diagrams, charts, and illustrations to instructional computing materials. PROGRAM 25 demonstrates the use of these statements to generate unique art. The program is designed to:

1. Clear the screen and color it light blue.
2. Draw 100 vertical and 100 horizontal lines of varying lengths at random locations on the screen.
3. Draw each line using a random color.

Since PROGRAM 25 is dynamic, it must be RUN to be appreciated. The actions on the screen cannot be sufficiently illustrated by words or pictures in this text.

Run from disk and refer to the listing of PROGRAM 25.

```
]LOAD PROGRAM 25
]LIST

100  REM  ===============
110  REM  PROGRAM 25 DESCRIPTION
120  REM  ===============
130  REM  DEMONSTRATION OF LOW-RESOLUTION
     GRAPHICS.
140  REM  THE SCREEN WILL BE COLORED BLUE AND
     100 RANDOM
150  REM  LINES OF RANDOM COLORS WILL BE DRAWN.
160  REM  ===============
170  REM  VARIABLE DICTIONARY
180  REM  ===============
190  REM  A - RANDOM STARTING POINT
200  REM  B - RANDOM ENDING POINT
210  REM  C - RANDOM X OR Y POINT
220  REM  I - LOOP COUNTER
230  REM  X - RANDOM COLOR CODE
240  REM  ===============
250  REM  COLOR IN BACKGROUND
260  REM  ===============
270  HOME
280  GR
290  COLOR = 7
300  FOR I = 0 TO 39
```

100–230 document program and list important variables and what they represent.

270 clears text page. (Undesired text may otherwise appear below graphics.)

280 initializes low-resolution graphics screen.

```
310  HLIN 0,39 AT I
320  NEXT I
330  REM  ===============
340  REM  LOOP 100 TIMES
350  REM  ===============
360  FOR I = 1 TO 100
370  REM  ===============
380  REM  CHOOSE RANDOM COLOR
390  REM  ===============
400  LET X =  INT( RND (1) * 16)
410  IF X = 7 THEN 400
420  COLOR = X
430  REM  ===============
440  REM  PLOT VERTICAL LINE
450  REM  ===============
460  GOSUB 610
470  VLIN A,B AT C
480  REM  ===============
490  REM  ANOTHER RANDOM COLOR
500  REM  ===============
510  LET X =  INT ( RND (1) * 16)
520  IF X = 7 THEN 510
530  COLOR = X
540  REM  ===============
550  REM  PLOT HORIZONTAL LINE
560  REM  ===============
570  GOSUB 610
580  HLIN A,B AT C
590  NEXT I
600  END
610  REM  ===============
620  REM  SUBROUTINE TO CHOOSE
630  REM  THREE RANDOM POINTS
640  REM  ===============
650  LET A =  INT ( RND (1) * 40)
660  LET B =  INT ( RND (1) * 40)
670  LET C =  INT ( RND (1) * 40)
680  RETURN
```

290 sets COLOR light blue; 300–320 draw 40 horizontal lines completely across screen (x-axis points 0–39). In effect, this colors background light blue.

360 defines loop which terminates at 590. It will be executed 100 times.

400 chooses random number 0–15.

410 checks if chosen number equals 7, the background color code. If so, another will be chosen. Otherwise, COLOR set to random number at 420.

460 calls subroutine at 610.

470 draws vertical line from A on y-axis to B on y-axis at point C on x-axis. (A, B, C, values come from subroutine at 610.)

510–530 select random COLOR not light blue. Execution transferred again to subroutine at 610; 3 new random numbers stored in A, B, C.

580 draws horizontal line from A on y-axis to B on x-axis at point C on y-axis.

590 terminates loop. Since loop executes 100 times, 100 random vertical and horizontal lines are drawn on screen.

610–680 is subroutine generating 3 random numbers (0–39). It stores them in A, B, C.

7.4 STATEMENTS FOR HIGH-RESOLUTION GRAPHICS

The Apple II microcomputer has two levels of graphics available: low-resolution graphics, as discussed above, and high-resolution graphics. High-resolution graphics, as the name implies, have greater detail or more resolution. However, something must be sacrificed for this feature—the variety of colors.

The high-resolution screen is illustrated in Figure 7.2. A point on the screen is specified in the same fashion as on the low-resolution screen by giving the x-axis position first, followed by the y-axis position. However, the axes have con-

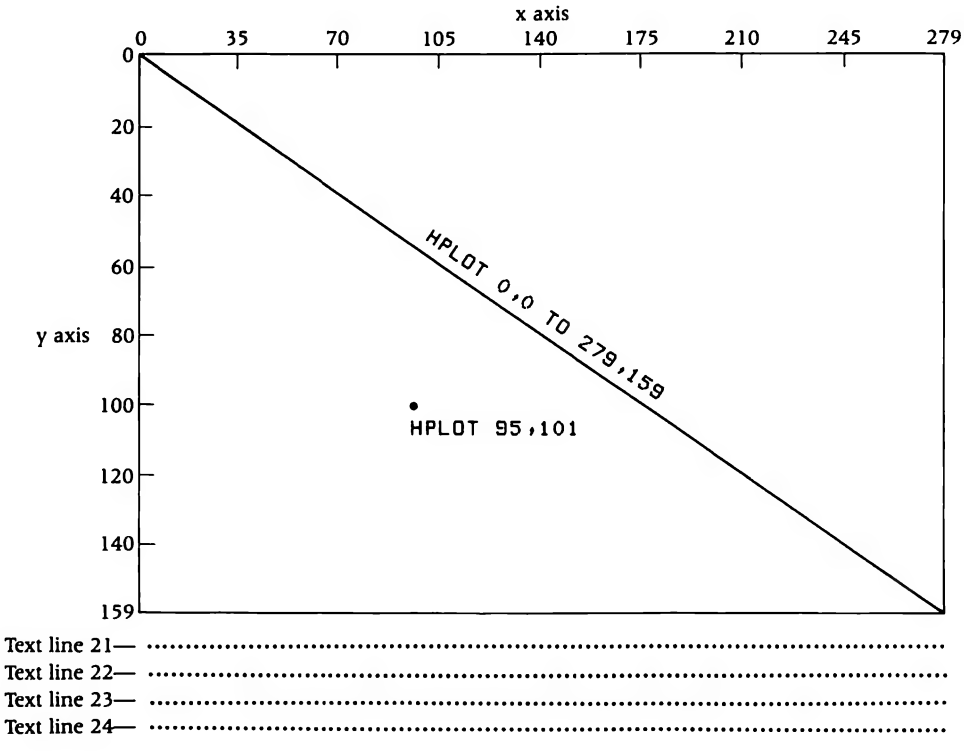


Figure 7.2
High-resolution graphics screen (HGR).

siderably more units: The x-axis contains 280 units (0 through 279) and the y-axis contains 160 (0 through 159). Four text lines are available at the bottom of the screen for instructions, questions, and comments.

Instead of having sixteen colors available, only six are allowed: black, white, green, blue, orange, and violet. These colors will vary in hue depending on the brand of TV monitor being used.

7.4.1 Statement HGR

Purpose The HGR statement is used to initialize the high-resolution graphics screen in a program. When it is executed, the computer monitor will change from text to high-resolution graphics, and the screen will be cleared to black. As pictured in Figure 7.2, the high-resolution screen initialized with the HGR statement contains 44,800 points (0 to 279 by 0 to 159).

7.4.2 Statement HCOLOR

Purpose The HCOLOR statement sets the color for subsequent graphics statements. Once the color has been set, all graphics drawn on the screen will be of that color until another HCOLOR statement is executed. Six colors are

available. Each color is represented by a number from 0 to 7 (black and white are represented by two codes):

```
HCOLOR = 0    (black)        HCOLOR = 4    (black)
HCOLOR = 1    (green)        HCOLOR = 5    (orange)
HCOLOR = 2    (violet)       HCOLOR = 6    (blue)
HCOLOR = 3    (white)        HCOLOR = 7    (white)
```

7.4.3 Statement HPLOT

Purpose HPLOT will place a dot on the screen at the x- and y-coordinates specified in the statement. The color of the dot will be the color specified by the most recently executed COLOR statement.

Example:
```
HPLOT 95,101
```

(Plots a dot on the high-resolution screen 95 units to the right on the x-axis and 101 units down the y-axis. See Figure 7.2.)

The HPLOT statement can also be used to draw a line from one point on the screen to another.

Example:
```
10 HGR
20 HCOLOR = 3
30 HPLOT 0,0 TO 279,159
40 END
```

(Draws a diagonal white line from the upper left corner to the lower right corner of the screen. See Figure 7.2.)

HPLOT can also be used to draw a line from the last point plotted to the x- and y-coordinates specified.

Example:
```
10 HGR
20 HCOLOR = 3
30 HPLOT 0,0
40 HPLOT TO 279,0
50 HPLOT TO 279,159
60 HPLOT TO 0,159
70 HPLOT TO 0,0
80 END
```

(Draws a white border completely around the high-resolution screen.)

A series of lines can be specified in a single HPLOT statement. The following example will have the same result as the previous example (a border around the graphics screen); however, it is done in one statement.

Example:
```
10 HGR
20 HCOLOR = 3
30 HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO
   0,0
40 END
```

7.4.4 Statement VTAB

Purpose The VTAB statement tabs to the line number specified so that text can be PRINTed on that line. Both the low-resolution and high-resolution graphics screens have four text lines available. These lines are the 21st, 22nd, 23rd, and 24th lines on the text screen. VTAB 21 in statement 80 of the following program allows statement 90 to PRINT on line 21 and statement 100 to PRINT on line 22.

Example: Enter the following high-resolution graphics program and RUN it:

```
NEW
5 HOME
10 HGR
20 FOR I = 1 TO 100
30 HCOLOR = INT(RND(1)*8)
40 X = INT(RND(1)*280)
50 Y = INT(RND(1)*160)
60 HPLOT X,Y
70 NEXT I
80 VTAB 21
90 PRINT "THE STARS AT NIGHT...ARE BIG AND
   BRIGHT"
100 PRINT "      DEEP IN THE HEART OF TEXAS.";
110 END
```

10 initializes high-resolution graphics screen.

20–70 defines loop executed 100 times.

30 chooses random color.

40 chooses random x-axis position.

50 chooses random y-axis position.

60 plots dot at chosen coordinate.

Brilliant! One hundred random points (stars) were plotted on the screen.

7.5 HIGH-RESOLUTION GRAPHICS AND INSTRUCTIONAL COMPUTING MATERIALS

When developing instructional computing materials that contain graphics, some special planning is necessary. In addition to the normal designing of the program, the graphic screens used in the program should be sketched or plotted on graph paper. Longer tutorial programs may require a *storyboard* to be prepared. This is a series of sketches of the graphics with the related textual information or questions included.

When designing the program, the graphics are most easily done in subroutines which can be called as needed in the program. The subroutines can be easily tested by typing RUN and the starting line number of the subroutine. (For example, RUN 800 would execute the subroutine beginning at line 800.)

7.5.1 PROGRAM 26: Shape-Recognition Drill

PROGRAM 26 is a drill-and-practice program that displays a shape on the screen for the student to identify. Four shapes are used: circle, rectangle, square,

and triangle. The program randomly presents five questions, presents the shape in random sizes, and keeps track of the student's score. The program elements required in the design are:

1. Instructions to the student.
2. A loop to:
   a. Choose one of the four shapes.
   b. Branch to the appropriate subroutine.
3. Four subroutines (circle, rectangle, square, and triangle) to:
   a. Choose a random height and width.
   b. Plot the shape, centered on the screen.
   c. Ask the student to identify the shape.
   d. Input the student's answer.
   e. Display whether the answer is right or wrong.
   f. Tally the correct answers.
4. Display the number of correct answers.
5. End the program.

Run from disk and refer to the listing of PROGRAM 26.

```
]LOAD PROGRAM 26
]LIST

100  REM  ===============
110  REM  PROGRAM 26 DESCRIPTION
120  REM  ===============
130  REM  SHAPE-RECOGNITION DRILL.
140  REM  PROGRAM DRAWS A SHAPE ON THE SCREEN
150  REM  AND ASKS USER TO IDENTIFY IT.
160  REM  SHAPES ARE: CIRCLE, RECTANGLE, SQUARE,
     AND
170  REM  TRIANGLE. SHAPES ARE DRAWN IN RANDOM
     SIZES.
180  REM  ===============
190  REM  VARIABLE DICTIONARY
200  REM  ===============
210  REM  ANS$ - USER'S RESPONSE
220  REM  C - NUMBER CORRECT
230  REM  H - RANDOM HEIGHT
240  REM  I - LOOP COUNTER
250  REM  J - LOOP COUNTER
260  REM  W - RANDOM WIDTH
270  REM  X - X AXIS POINT
280  REM  Y - Y AXIS POINT
290  REM  Z - RANDOM SHAPE
300  REM  ===============
```

100–290 document program and list and identify variables.

```
310  REM  PRINT INTRODUCTION
320  REM  ===============
330  HOME
340  PRINT
350  PRINT "I AM GOING TO SHOW YOU SOME SHAPES."
360  PRINT
370  PRINT "YOU TELL ME WHAT KIND OF SHAPE IT IS."
380  PRINT
390  PRINT
400  INPUT "ARE YOU READY? ";ANS$
410  IF ANS$ <  > "YES" THEN 330
420  LET C = 0
430  REM  ===============
440  REM  ASK 5 QUESTIONS
450  REM  ===============
460  FOR I = 1 TO 5
470  HOME
480  HGR
490  VTAB 22
500  PRINT "C=CIRCLE R=RECTANGLE S=SQUARE
     T=TRIANGLE"
510  PRINT
520  REM  ===============
530  REM  CHOOSE RANDOM SHAPE.
540  REM  BRANCH TO SUBROUTINE.
550  REM  ===============
560  LET Z =  INT ( RND (1) * 4 + 1)
570  ON Z GOSUB 650,770,890,1010
580  NEXT I
590  HOME
600  TEXT
610  PRINT "YOU GOT ";C;" SHAPES CORRECT!"
620  PRINT
630  PRINT "SO LONG FOR NOW."
640  END
650  REM   ======
660  REM   SQUARE
670  REM   ======
680  LET H =  INT ( RND (1) * 61 + 10)
690  LET W = H * 1.20
700  LET Y = 80 - H / 2
710  LET X = 140 - W / 2
720  HPLOT X,Y TO X + W,Y TO X + W,Y + H TO X,Y
     + H TO X,Y
725  HPLOT X - 1, Y - 1 TO X + W+1, Y - 1 TO X
     + W + 1, Y + H + 1 TO X - 1, Y + H + 1
     TO X - 1, Y - 1
730  INPUT "WHICH SHAPE IS IT? ";ANS$
740  IF ANS$ = "S" THEN GOSUB 1170
750  IF ANS$ = <  > "S" THEN GOSUB 1260
```

330–410 clear screen, provide basic instructions, and ask student if ready. If so, program continues at 420; if student not ready, instructions are repeated.

420 sets correct-answer counter, C, to zero.

460 begins loop terminating at 580. It is executed 5 times.

470–480 clear text screen and initialize high-resolution graphics screen.

490–510 print answer codes on text line 22.

560 chooses random number (1–4).

570 branches to corresponding subroutine.

590 clears screen after loop executed 5 times.

600 switches back to full text screen.

610 reports number of correct responses.

630 makes concluding remarks.

650–760 is subroutine that draws a square.

680 generates random height for square.

690 multiplies height by 1.2 to obtain width of square. (Because screen is rectangular, 1 unit on x-axis ≈ 1.2 units on y-axis.)

700–710 calculate the starting y- and x-axis positions; respectively, to center square on screen. (*Note:* 140,80 is approx. center screen.)

720–725 draw the square.

730 prints question on text line 24 and inputs answer into ANS$. If answer is "S" for square, subroutine at 1170 is executed. Otherwise, subroutine at 1260 is executed.

```
760  RETURN
770  REM  ========
780  REM  TRIANGLE
790  REM  ========
800  LET H =  INT ( RND (1) * 61 + 10)
810  LET W = H * .7
820  LET Y = 80 - H / 2
830  LET X = 140
840  HPLOT X,Y TO X + W,Y + H TO X - W,Y + H
     TO X,Y
850  INPUT "WHICH SHAPE IS IT? ";ANS$
860  IF ANS$ = "T" THEN GOSUB 1170
870  IF ANS$ <  > "T" THEN GOSUB 1260
880  RETURN
890  REM  =========
900  REM  RECTANGLE
910  REM  =========
920  LET H =  INT ( RND (1) * 61 + 10)
930  LET W = H * 2
940  LET Y = 80 - H / 2
950  LET X = 140 - W / 2
960  HPLOT X,Y TO X + W,Y TO X + W,Y + H TO X,Y
     + H TO X,Y
965  HPLOT X - 1, Y - 1 TO X + W+1, Y - 1 TO
     X + W + 1, Y + H + 1 TO X - 1, Y + H + 1 TO
     X - 1, Y - 1
970  INPUT "WHICH SHAPE IS IT? ";ANS$
980  IF ANS$ = "R" THEN GOSUB 1170
990  IF ANS$ <  > "R" THEN GOSUB 1260
1000  RETURN
1010  REM  ======
1020  REM  CIRCLE
1030  REM  ======
1040  LET H =  INT ( RND (1) * 61 + 10)
1050  LET X =  COS ( - 3.14) * H * 1.2 + 140
1060  LET Y =  SIN ( - 3.14) * H + 80
1070  HPLOT X,Y
1080  FOR J = -  3.15 TO 3.15 STEP .1
1090  LET X =  COS (J) * H * 1.2 + 140
1100  LET Y =  SIN (J) * H + 80
1110  HPLOT  TO X,Y
1120  NEXT J
1130  INPUT "WHICH SHAPE IS IT? ";ANS$
1140  IF ANS$ = "C" THEN GOSUB 1170
1150  IF ANS$ <  > "C" THEN GOSUB 1260
1160  RETURN
1170  REM   ===============
1180  REM   ANSWER CORRECT
1190  REM   ===============
1200  PRINT
1210  PRINT "YOU ARE CORRECT!"
```

1170–1250 is subroutine that informs student of correct answer (1210) and adds 1 to correct-answer counter C (1220).

1230 and 1240 loop 1000 times to slow drill to pleasing pace.

1260–1330 is subroutine that informs student of wrong answer (1300).

1310 and 1320 also loop 1000 times to slow pace.

770–880 and 890–1000 are subroutines to draw triangle and rectangle. They follow same logic as square. Study them to discover technique used in each.

1010–1160 is subroutine that draws a circle. Rather than connecting corners of an object, as with square, triangle, and rectangle, circle must be drawn by computing each x- and y-coordinate from a formula (polar coordinate formula).

1040 randomly chooses value for circle radius.

1050 calculates x-axis coordinate for starting point; 1060 calculates y-axis coordinate. Formulae are:
X = COS(radian) × (radius) × (1.2)
+ (x-coordinate for center)
Y = SIN(radian) × (radius)
+ (y-coordinate for center)

1050 substitutes −3.14 for radian, variable H for radius, and 140 for x-coordinate of circle center. Remember, 1.2 is width-to-height ratio of screen.

1060 substitutes −3.14 and H, and 80 for y-coordinate of circle center.

1070 plots starting point on screen.

1080–1120 calculate subsequent x- and y-coordinates around circle. Loop runs from −3.15 to +3.15 (circle contains 2π radians), stepping by 0.1. This stepping factor provides a relatively smooth circle; yet it plots at reasonable speed.

```
1220  LET C = C + 1
1230  FOR J = 1 TO 1000
1240  NEXT J
1250  RETURN
1260  REM   ===============
1270  REM   ANSWER WRONG
1280  REM   ===============
1290  PRINT
1300  PRINT "SORRY, TRY ANOTHER."
1310  FOR J = 1 TO 1000
1320  NEXT J
1330  RETURN
```

1110 draws line from previous point plotted to current values of X and Y.

1130–1150 ask student to identify shape and branch to appropriate subroutine if answer right or wrong.

7.6 SOME NOTES ABOUT USING COLOR

The graphic statements in this chapter can be employed to "add a little color" to instructional computing materials. However, there are both positive and negative factors to be considered when using color:

1. Color can increase attention.
2. Color can increase motivation.
3. Color is less fatiguing to the eye than black and white text.
4. If color is used for highlighting concepts, it must be used consistently throughout the program.
5. Limit the number of colors used at any one time to four.
6. Use highly saturated (bold) colors.
7. Consider color stereotypes. (Stop signs must be red.)
8. The greater the contrast between two colors (i.e., complementary colors), the greater the visual impact.
9. Remember that 10% of all males and 5% of all females are color-blind.
10. *Most important:* If you emphasize everything, nothing on the screen will stand out!

7.7 POSERS AND PROBLEMS

1. Correct any errors in the following statements:

```
10 GR
20 COLOR = 10
30 HPLOT 10,10 TO 100,100
40 END
```

2. Modify PROGRAM 25 to draw random squares of random colors on the low-resolution graphics screen instead of random lines.

3. What would result from the execution of the following statements?

```
10 HGR
20 HCOLOR = 2
30 FOR Y = 0 TO 159
40 HPLOT 0,Y TO 279,Y
50 NEXT Y
60 END
```

4. Write a low-resolution graphics program that displays sixteen bars of different colors. (This program can be used as a test pattern to adjust the color on the TV monitor.)

5. Write a low-resolution graphics program which displays a checkerboard pattern (your choice of colors) on the screen.

6. Write a high-resolution graphics program that plots the function X = SQR(Y)*20 (vary Y from 0 to 159).

Part II

An Introduction to the Design and Development of Instructional Computing Materials

"It takes less time to do a thing right than to explain why you did it wrong."
—H. W. Longfellow

"Garbage in, garbage out."
—Anon.

"A thing of beauty is a joy forever."
—John Keats

Think About This (for Fun)

Using each number only once, arrange the figures 0,1,2,3,4,5,6,7,8,9 so that their sum is 100.

Think About This (Seriously)

Should every student have had an exposure to computers and their uses by the time of graduation from high school?

Chapter

8

What Are Your Intentions?

8.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Identify the steps of a "systems approach" to the design of instructional computing materials (Section 8.3).
2. Identify an area of personal interest within which to apply instructional computing.
3. Outline a rationale, a set of quantitative performance objectives, and a sequence of instruction for a unit of instructional computing materials (Sections 8.3.1–8.3.3).

8.2 DESIGNING INSTRUCTIONAL COMPUTING MATERIALS

A working knowledge of BASIC (or any programming language) provides only a very small step toward the actual development of educationally valid instructional computing materials. In fact, such materials have been designed by educators with no computing experience whatsoever! In these cases, the completed design is given to a computer programmer (who often knows very little about the specific academic area) for translation into an executable computer program. The executable program is tested, refined, and eventually put to use in the classroom. Thus, the key to the development of valid educational materials rests initially with their *design*.

The entire design and development process can be improved if both the author and the programmer have something more than a casual awareness of the other's area of expertise. However, it is not often that the author and programmer are one and the same person, with expertise in both programming and a given academic area. Very few educators have high proficiency in programming techniques and strategies. Likewise, few programmers know the intricacies of learning theory, instructional design, teaching methodology, and so on.

The wide acceptance and use of microcomputers in education is bringing about a gradual change in this, however. More and more, both inservice and preservice teachers are gaining knowledge in computer literacy and instructional computing uses. With this knowledge will come improved materials and improved use of this medium of instructional technology which, literally, is at our fingertips.

Design! It is not too unusual for some individuals to have the feeling that they have never designed anything! However, if they have ever wanted anything, anything at all, that was eventually obtained through their efforts, they have experienced the design process! This process, then, is really something common to most people, and it has at least one fringe benefit: It makes us think logically and creatively. That is, the procedure—from identification of an objective to its attainment—becomes a series of steps.

Often, this logical procedure is called an *algorithm*, and, in fact, it *is* a logical series of steps that must be followed in designing *any* effective package of instructional materials. This process, however, is amplified greatly in designing and developing interactive instructional computing materials. There are several reasons for this amplification, the primary ones being the immediate feedback and active user participation aspects of instructional computing. The design of a program—for better or worse—rapidly becomes apparent to a user through the interactive nature of this type of instructional media.

8.3 THE SYSTEMS APPROACH

The design stage of instructional computing materials is one part of a process that is used extensively in the overall development of educational materials.

Although this process is known by several names, and the steps may differ slightly among versions, it may be summarized as follows:

1. statement of the rationale for use
2. statement of quantitative performance objectives
3. definition of the instructional sequence
4. program construction
5. debugging
6. pilot testing
7. revision
8. use in the classroom
9. revision
10. evaluation

These ten steps comprise the process often called *a systems approach to instructional design*. However, since it does involve a logical approach, another name might be, "A Common Sense Approach to Instructional Design."

The first three steps constitute the design stage and will be discussed in this chapter. The following seven steps will be discussed in Chapter 9. Note that, although all of these steps are important, the contents of each are determined solely by the author(s) of the instructional computing materials. In other words, the steps and general procedures for each can be outlined in this book, but the reason for any given instructional computing lesson—what it does and how it does it—can only be determined by its author(s).

8.3.1 The Rationale

Assume that an area of interest has been identified for the design and development of a unit of instructional computing material. Can reasons be stated why this particular area of interest should be taught in the first place? Can reasons be stated why a computer should be used? In other words, the rationale is the answer to *why*: *Why* teach this academic concept, and *why* use the computer as an adjunct to the instructional process? If the *why* cannot be justified in both instances, the design stage should be terminated and another area of interest identified.

The following examples of rationales are taken directly from instructional computing units developed by various teachers. Note how brief or how thorough such a rationale may be. The first example is very brief:

> The purpose of this learning module (unit) is to enrich the student's personal communication skills, provide a background knowledge for future study in business and

economics, and provide a beginning knowledge base of terminology for application in the selected career area. Terminology is essential for communicating in a specialized technological society. This module provides a beginning for building a vocabulary base in business, management and economics.

A second, slightly longer rationale is very specific:

Correct association of compound names with molecular formulas is a necessary skill for continuing successfully in a chemistry course. The names and formulas for compounds are used interchangeably throughout most chemical literature. Mastery of chemistry textbook reading material requires the correct identification of compound names and formulas. In the chemistry laboratory, names and formulas are also used interchangeably in labeling containers and in written laboratory procedures. A serious error could result in the laboratory if a student incorrectly identified a compound used in the experiment.

The computer can serve as an effective tool for the student who is learning to identify the names and molecular formulas of compounds because: 1) it allows the student to work at his/her individual pace, 2) it provides immediate feedback to the student after each answer is given, 3) it may randomly generate different questions so that the student has a variety of practice, 4) it scores the student at the end of the drill providing an estimation of progress, and, 5) it may be adapted for use in both drill exercises and testing.

The third example is as specific as the second and is slightly more expansive:

Preservice educational preparation for nursing in a coronary care unit generally focuses on dysrhythmia recognition. Given various electrocardiographic tracings, the learner is expected to label the patterns by origin and conduction of impulse, rate, and probable clinical sequela. She/he is rarely provided opportunity to project and evaluate nursing actions based on recognition of the dysrhythmia. Consequently, these decision making skills are usually learned "on the job" under tutelage of a more experienced nurse practitioner. The trainee's learning depends, then, on numerous variables—the experienced nurse's willingness to teach, clinical situations which "happen" to be present, critical time factors which may or may not permit the trainee opportunity to project appropriate actions before action is required, and numerous other equally uncontrollable factors. Preservice teaching methods can, and should, be developed which facilitate the trainee's acquisition of decision making/judgment skills in environments created deliberately for learning; learning within the setting of a coronary care unit is best reserved for only those abilities which cannot be synthesized in any other environment.

Simulation is one possibly effective preservice teaching technique to facilitate acquisition of decision making/judgment skills. Simulation teaching strategies have been noted to enable the student to: 1) actively participate in learning, 2) integrate theoretical concepts to simulated life situations, 3) desensitize oneself against threatening situations, 4) be presented with identical "hands-on experiences" as those presented fellow learners, 5) experience some of the doubts, competencies, difficulties and anxieties that would be experienced in actual clinical settings, and, 6) respond in a

safe standardized context free of concern about harming the patient or pleasing a tutor.

What are the advantages of using the computer in designing these simulated experiences? First, the selection and sequencing of problems can be randomized independent of instructor or learner choice at the moment—a situation more closely approximating the "randomness" of the actual clinical setting. Second, the learner can be provided with immediate feedback on decisions made. Third, since computers are interactive, the student's response has a measurable effect on the material as it is presented. Fifth, the learner can choose the time for instruction, times when faculty may or may not be available. Sixth, the instructor can reconstruct precisely the sequence in which the student responds to the simulated clinical situation, diagnose errors in approach, and pinpoint reinforcement and help.

In summary, the rationale underlying this unit rests on three premises: 1) A need for preservice acquisition of decision making/judgment skills exists. 2) Simulated experiences can assist in acquisition of these needed skills. 3) Use of the computer enhances the student's independence, assists instructor diagnosis of learning difficulties, and facilitates the process of simulating clinical situations.

8.3.2 Quantitative Performance Objectives

Students will be interacting with your programs: Do they know what is expected of them before, during, and after this interaction? Before a student sits down at a computer terminal, information should be provided that at least outlines the prerequisites for interaction, what the interaction will deal with, and, *specifically*, what constitutes a successful interaction. For what goals should the student strive, and how will it be determined if these goals are attained?

Continuing with our examples from the previous section, a statement of quantitative objectives might be as brief as:

General: Given a basic list of business terms, the student will develop a working knowledge of basic business terms. The student will demonstrate this ability by completing successfully the instructional computing units focusing on terminology mastery.

Specific: Given a set of terminology, the student will complete the instructional computing unit with 90% or better accuracy on a 20-word list.

The second example is succinct and equally brief:

1. The student will be able to state the name of a compound when given its molecular formula with 80% accuracy.
2. The student will be able to state the molecular formula of a compound when given its name with 80% accuracy.

The third example is longer but also quite specific:

- Given a cardiac rhythm strip, the student will identify the pattern by site of origin and rhythm with 100% accuracy.
- Given a cardiac rhythm strip, the student will identify an appropriate sequence of nursing actions from among the following four alternatives: obtain more data, execute a standing order, call the physician, or continue close observation.
- Given a decision to call the physician, the student will indicate the information to be shared, omitting no pertinent data.
- Given a decision to obtain more data, the student will ask for data pertinent to formulating a subsequent action-decision.
- Given feedback regarding a questionable action-decision, the student will re-evaluate the decision and indicate with 100% accuracy if the decision was appropriate.

For a thorough and enlightening description of defining instructional objectives, the reader is referred to the classic text in this field, *Preparing Instructional Objectives* by R. F. Mager (Fearon Publications, Palo Alto, Calif., 1962).

8.3.3 The Instructional Sequence

This step in design is probably the most difficult for tutorial dialog programs and the least difficult for linear (nonbranching) programs. Obviously, the instructional sequence is in part determined by the type of instructional computing (problem solving, drill, simulation, etc.) to be applied. This in turn is determined by the rationale, objectives, and interactive tasks defined for the unit. Regardless of the type of use, this step should include, as a minimum, answers to such questions as:

1. Should review material or other information specifically related to the unit be provided prior to actual interaction? If so, what?
2. What student-control options should be included? Stop at will? Skip problems or sections? Receive answers to questions without an actual attempt at answering?
3. How many questions will be included in the interaction?
4. What are the anticipated correct answers to questions? What response(s) will be given?
5. What are the anticipated incorrect answers to questions? What response(s) will be given?
6. What will the program do if neither an anticipated correct nor incorrect answer is matched? Give a hint? Give the answer?
7. How many "misses" will be allowed?
8. Will branching to review sections be provided for students having difficulty?

9. Will only answers that are correct on the first attempt be recorded?
10. How will the performance report to the student be presented? Will areas of strength and/or weakness be identified?

Answers to these—and perhaps many other questions, depending upon the design—must be outlined on paper prior to translation of the defined sequence into a computer programming language.

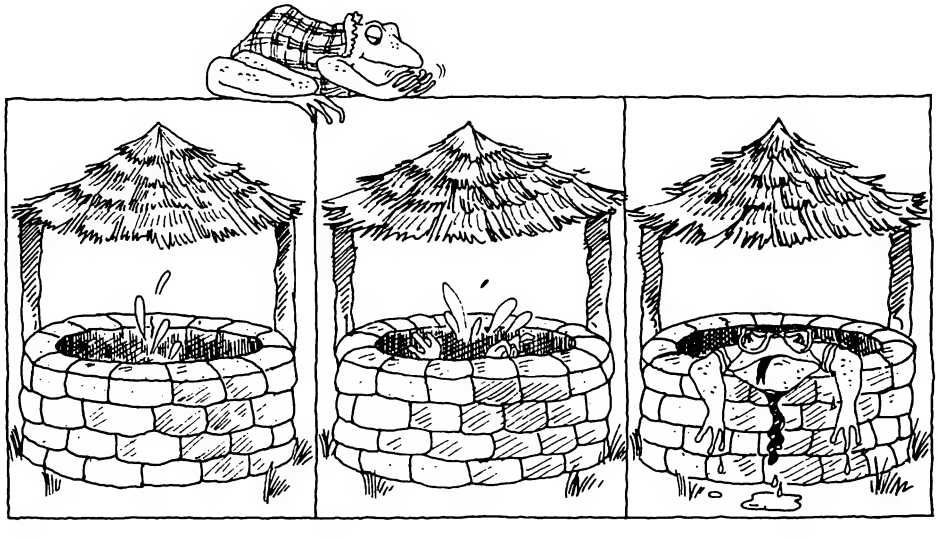
8.4 POSERS AND PROBLEMS

1. Outline on paper the rationale, quantitative objective(s), and sequence of instruction for a short unit of instructional computing in an area of your interest.

"The young do not know enough to be prudent and therefore they attempt the impossible—and achieve it, generation after generation."
—Pearl S. Buck

"The next-best thing to knowing something is knowing where to find it."
—The Ensign

"Them as has, gits."
—Anon.



Think About This (for Fun)

A frog (male) is at the bottom of a thirty-foot well, trying to escape. Everytime he jumps up three feet, he falls back two. How many jumps will it require for the frog to get out?

Think About This (Seriously)

Should our society become a computer-literate society? If so, how could this be accomplished?

Chapter

9

Developmental Processes

9.1 OBJECTIVES

For the successful completion of this chapter, you should be able to:

1. Identify the processes involved in the developmental steps of the systems approach to instructional design (Section 9.2).
2. Identify at least ten of the twelve guidelines for the design and development of instructional computing materials (Section 9.3).
3. Using information discussed in Chapters 1 through 9, design and develop instructional computing units.

9.2 THE SYSTEMS APPROACH (continued)

The design of instructional computing materials constitutes the first three steps of the systems approach. These steps are essentially mental, paper-and-pencil

processes. Once the rationale, objectives, and instructional sequence have been defined, the remaining steps of the development process—the coding, debugging, testing, refinement, and use and evaluation of the materials—may be started.

The total process, from rationale to evaluation, for an original set of instructional computing materials may require 50 to 250 person-hours for each hour of student interaction at a terminal. This would include development of any accompanying materials, such as student and instructor manuals. Of course, if model programs are simply adapted to a teacher's specific needs, the time required for development is considerably reduced.

9.2.1 Program Construction

Actually, this step is still a mental, paper-and-pencil process for the most part. It primarily involves the translation of the instructional sequence into computer program statements. This is the first of the systematic steps in which some degree of programming expertise is required from either the design author or a programmer. Programming techniques and strategies must be used in transferring the design concepts from paper to executable program code. This step may range from the trivial task of adapting a model program to the extremely involved, time-consuming process of translating an original, detailed design into program code.

9.2.2 Debugging

Once the code has been written, entered, and saved, execution of the program is attempted. Chances are, the program will not run. Problems, commonly called *bugs* in computerese, may be present. These may be anything from simple syntax errors (omitting quotes, misspelling statements, etc.) to technical or conceptual errors (incorrect use of a formula, right answer not accepted, omitting counters, branching at the wrong point, etc.). *Debugging* (extermination of the errors) is done to the point that program execution is satisfactory from the author's viewpoint.

9.2.3 Pilot Testing

Pilot testing of the program is performed next. Generally, this is done with the aid of teaching colleagues and a few volunteer students to test the program on an individual basis. It is recommended that the author literally "look over their shoulders" as they run the program since it is a rare case in which something unanticipated does not occur. These events may be as trivial as the user typing in an anticipated answer, followed by an unanticipated period or space which the program cannot handle. Alternatively, a major discussion of the conceptual and/or instructional strategy may be involved. Of course, the main point of pilot testing is *feedback* to the author regarding the design and content of the program.

9.2.4 Revision

It is common for instructional computing materials to be frequently revised. However, the majority of revisions occur after pilot testing. These revisions are usually fairly minor in nature, involving redefining anticipated answers, improving responses, making cosmetic improvements to the display, and so on. However, the revisions could be as major as returning to the design stage for refinement of the program or, in extreme cases, discarding the program. (If the design steps are thought out carefully, this probably will not occur!) Note that the pilot testing and revision steps are cyclic and may be repeated several times prior to actual classroom use of the program.

9.2.5 Use in the Classroom/Further Revision

Use of instructional computing materials in the classroom is, obviously, directly related to the design of the materials. This use may be supplemental for those students needing review or assistance on a given concept; it may be a required segment of a set of "learning activities"; it may be a prerequisite simulation of a real experiment prior to entering the laboratory; it may be used both as a drill and a testing procedure; and so on.

Regardless of the particular application, it is safe to anticipate minor revision of the materials, if for no other reason than the number of users testing the materials will have increased. Again, it is unlikely that the materials will ever get to the point where no additional revisions (however minor) are needed. Thus, use in the classroom and revision are cyclic and may continue as long as the materials are a part of the given instructional process.

9.2.6 Evaluation

Evaluation of instructional computing materials may be divided into two categories. The first is an analysis to determine if the students are indeed attaining the defined objectives. This analysis may vary depending upon the design of the materials, but it is often based upon pretest and posttest results. If negative results are indicated, a return to Step 1 of the systems approach may be appropriate.

The second evaluation is of the *concept* of using instructional computing materials. Did this approach as an instructional medium prove suitable? Analysis of this comes in part from evaluation of the materials in terms of meeting defined objectives. Further evaluation may be based on both student and colleague feedback via attitudinal questionnaires, overall student performance, and, although it lacks quantitative measurement, the author's intuitive feeling.

Note: Research since the late sixties has consistently indicated that the concept of the use of supplemental instructional computing materials is educationally valid. In general, the success or failure of any given instructional computing program rests heavily upon the design steps previously discussed. Although it should go without saying, the importance of thoughtful design merits emphasis

one final time. If, in particular, the rationale, objectives, and instructional sequence are very carefully defined, the chances for successful use of the materials are greatly enhanced. In other words, think it through, folks!

9.3 GUIDELINES FOR DESIGN AND DEVELOPMENT

9.3.1 Consider BASIC

Although there are some disadvantages to using BASIC as an instructional computing language (primarily in translating instructional sequence into program code), they are minor when compared to the relative ease of acquiring a working knowledge of the language, its universal nature, and its transportability.

9.3.2 Modularize the Units

It is good practice when writing any computer program to keep it as *modular* (concise by topic) as possible. For example, if a given concept includes a series of subconcepts, it is better to have one program for each subconcept, rather than one long program for the total concept. Programs are not only easier to design on this basis but are also easier to debug and revise.

9.3.3 Follow a Systems Approach

It is obviously important that the author of a program know the why, what, how, and effect of using instructional computing materials. For purposes of motivation, it is equally important that the student know why the area is worth studying, what the objectives are, how they will be achieved, and what effect they will have. Following a systems approach in the design and development of the materials is a means by which this may be accomplished.

9.3.4 State Quantitative Objectives

Although this is one of the steps in the systems approach to instructional design, it merits reiteration. Ensure that users of instructional computing materials know specifically the extent and effect of a successful interaction with the materials. This means that measurement of the objectives must be possible.

9.3.5 Put in Personality

Be kind to the users of your materials. Have a variety of positive reinforcers. Avoid the use of any negative feedback to the student; rather, make your responses to incorrect answers indicate that you are there "in spirit" to assist the student, and then proceed to do so. Include enough humor to solicit a smile or two from the user, but avoid the use of "cute" statements and repetitive responses. Also avoid the use of "fad" responses; they go out of style quickly.

9.3.6 Consider Gluteal Limits

Another advantage of modularization is that the user will not be sitting at a terminal for lengthy periods. A good "rule of rear" is to keep the interaction to 30 minutes or less.

9.3.7 Avoid Lengthy Text

Do not make programs "page turners"! It is expensive and boring. One of the key elements in successful instructional computing is that the user be an *active* learner. If detailed information, figures, tables, and so on, are required, have these available as supplemental materials prior to or during the interaction.

9.3.8 Branch

Another key to success is the individualization that may be provided by branching. If appropriate, the program should have the capability to allow students to view additional material, skip areas if competence is indicated, and/or stop the interaction at will, based upon student need or performance. In any event, never construct a program so that the student is trapped in a routine with no means of escape. Always provide some means by which the student may continue. For example, give the answer after a certain number of incorrect responses or provide other options.

9.3.9 Supplemental Use

For better or worse, the major use of instructional computing is as a supplement or adjunct to traditional instruction. There are few courses that are taught by computer alone. Design units that will ease those areas that are routine to the instructional process or that can be best done by instructional computing techniques. Remember, it takes *teachers* to truly impart personality, lead discussions, and explain abstract concepts.

9.3.10 Document

Your work in the design and development of materials represents much time, effort, and thought. Thus, have your programs well documented with REMark statements and develop student and teacher guides where appropriate. This will facilitate not only the local use of your materials, but also their potential use elsewhere.

9.3.11 Review the Literature

Have others done what you are doing? Is their approach different from yours? Are you "reinventing the wheel"? Before you invest the effort required to design and develop materials, you should know what has gone before. Likewise, if your

work is unique and successful, consider publishing a description of what you have done. There are a variety of instructional computing journals and other publications available (see Appendix D). Others interested in instructional computing should have the opportunity to become aware of your efforts.

9.3.12 Recognize the Capabilities of the Computer

Finally, but perhaps foremost, never forget that, to this point in the realm of instructional computing, computers are an incredibly fast, accurate, and useful *tool*. They can only do what they have been programmed to do. That means that *people* are providing the instructions. Thus, computer programs are only as good or bad in their actions as they have been designed to be by the people who provided the instructions.

Instructional computing materials have been used successfully in problem solving, drill, testing, simulation, and, to a lesser degree, tutorial applications. In general, these are applications where speed and accuracy are important in improving the instructional process. That is where we are today.

Where will instructional computing be in the future? More and better of the same? Faster and cheaper computing? Computers in every home and school? Libraries of validated instructional computing materials? Use in practically every academic discipline? It is difficult to accurately predict this future, for the limits are determined by something unpredictable and unlimited: *imagination*.

Appendix

A

The Apple Computer and How to Use It

A.1 THE APPLE II COMPUTER

The Apple II microcomputer is one of the most popular computers used in education. Among the reasons for this popularity is its flexibility and expandability. An Apple owner can begin with a modest investment and gradually upgrade the system as his or her interest and budget allow.

The variety of components available for the Apple make it difficult to describe all the possible combinations. Therefore, this book will limit the discussion to the typical system found in schools:

1. Apple II Plus with 48K of RAM.
2. Color television or monitor.
3. Disk II floppy disk drive.
4. Dot matrix printer.

A.1.1 The "Core" of the Apple

From the exterior, the Apple resembles a typewriter with a keyboard but no place to put the paper. Inside the case of the Apple are the integrated circuits known as *IC's* or *chips* that make it operate. Figure A.1 illustrates the "core" of the Apple.

The functional work unit is the microprocessor chip which is located centrally in the computer. Surrounding the microprocessor are memory chips, peripheral slots, and other electronics necessary for the operation of the Apple.

Two types of memory are found in most microcomputers. ROM, *Read-Only Memory*, has programs already stored in it by the manufacturer. These programs may be read but not changed in any way. They are permanent and are never lost, even when the power is turned off. In contrast, RAM, *Random Access Memory*, is read-and-write memory. It may be read or changed (written to). When the power is turned off, any programs or data stored in RAM are erased.

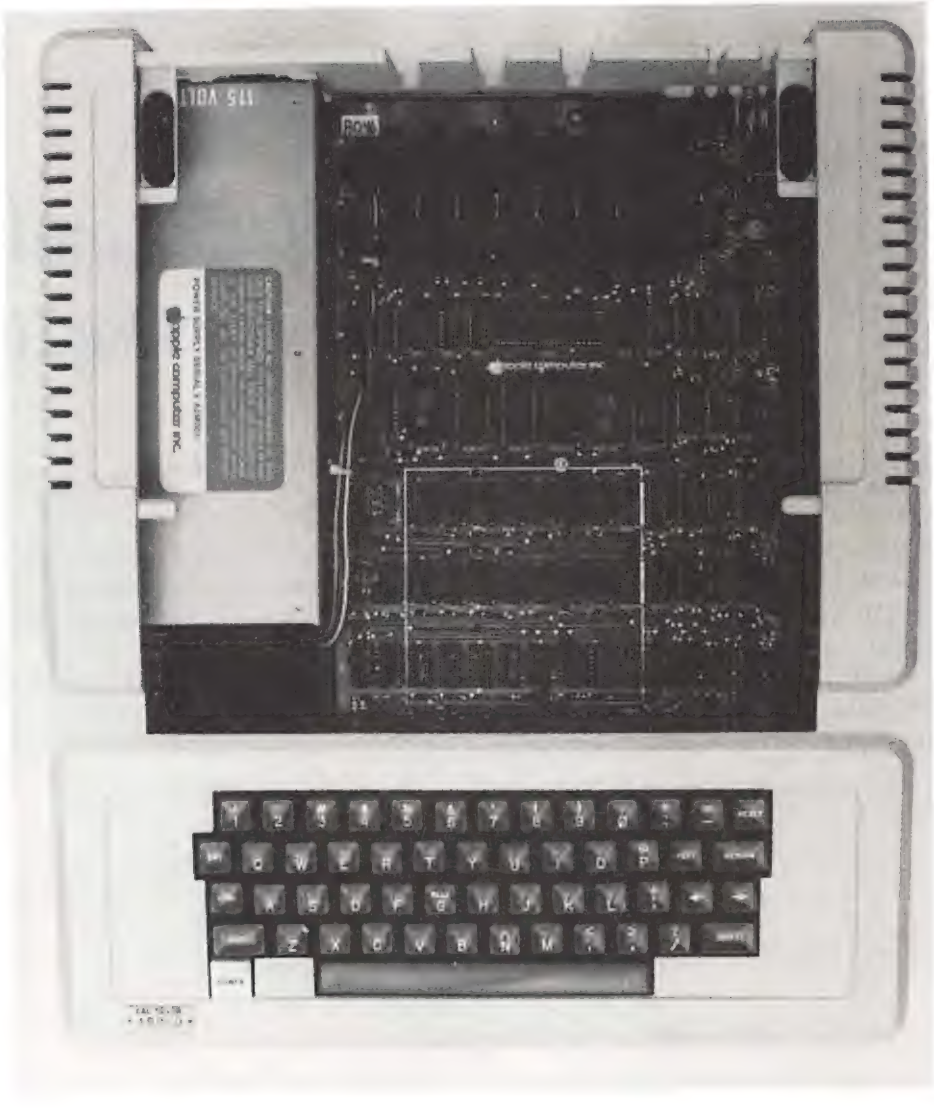


Figure A.1
The "core" of the Apple II. (Photograph by Carey Van Loon)

In the Apple II Plus, ROM contains the programs that make the computer operate (the *operating system*) and the Applesoft language interpreter. The latter will convert Applesoft BASIC statements and commands to meaningful codes to which the microprocessor can react.

In the Apple II, the predecessor of the Apple II Plus, ROM contained the operating system and the Integer BASIC language. If the reader wishes to use such a system with this text, he or she will need either the Applesoft Firmware card, which contains the same ROM as the Apple II Plus, or the Language System which contains 16K of RAM. The Language System works by loading the Applesoft BASIC interpreter into its RAM from the disk drive. (Note that, although the

Applesoft language can be loaded into RAM on the Apple II, it will not allow user access to high-resolution graphics and some of the programs contained in this book will not function properly.)

The Apple II Plus is available with 16K (16,384 characters of storage), 32K, or 48K of RAM. This memory is used to store a BASIC program, the program variables, the images of the text screen, the low-resolution graphics screen, and the high-resolution graphics screen. When using a disk drive, the Disk Operating System (DOS) containing the instructions to transfer data and programs between the Apple and the drive is loaded into RAM. This requires at least a 32K Apple system. If the user also wishes to utilize the high-resolution graphics screen in addition to a disk drive, a 48K Apple system will be needed. Consequently, most educators choose the 48K system.

Eight slots are provided inside the Apple toward the back. These slots are numbered 0 through 7 and are used to connect the Apple with peripheral devices. Slot 0, however, is the exception. It is used only for memory expansion and can contain the Applesoft Firmware or Language System cards mentioned above. Slots 1 through 7 are used for communicating with external devices such as printers (usually slot 1), other computers (slot 2), and disk drives (slot 6). Other less common peripherals include graphics tablet, clock, voice synthesis, voice recognition, plotter, and music synthesis.

The remaining integrated circuits in the Apple's core are used to generate the screen display, decode the keyboard input, and create sounds on the Apple's speaker. As with all electronic appliances, severe damage or shock can result from liquids being spilled inside the Apple. Appropriate care should be exercised.

A.1.2 The Television (Monitor)

The Apple II will output to any black and white or color television. (Of course, color graphics cannot be displayed in color on a black and white TV. Alternatively, either a black and white or color monitor can be used. A monitor will generally produce a sharper picture than a television; however, it is usually more expensive. The TV set is connected to the Apple with an RF modulator which converts the Apple's video signal to a TV signal. The modulator is connected from inside the Apple to the TV antenna leads. If a monitor is used, it is connected directly to the video output plug at the right rear corner of the Apple.

A.1.3 The Disk II Drive

The Disk II floppy disk drive is the "file cabinet" of the Apple. It is capable of storing 143,360 characters of information (programs and/or data) per diskette and can retrieve a single piece of information in 5/100000 of a second. The disk drive is connected to the Apple through an interface called a *disk controller* which is plugged into slot 6 of the Apple. Two drives can be connected to one controller, in which case they are usually labeled drive 1 and drive 2. This book utilizes only drive 1.

A.1.4 The Dot Matrix Printer

A variety of printers can be connected to the Apple through an interface plugged into slot 1. The most common and least expensive printer uses a pattern of dots to print the characters on the paper; hence the name *dot matrix printer.* The cost of printers range from approximately $400 to several thousand dollars; hence they are considered by some to be a "luxury" in the educational setting. However, a printer is essential to the process of developing instructional computing materials.

A.2 HOW TO USE THE APPLE WITH THIS BOOK

A companion to this book is a diskette containing all of the sample programs described in the various chapters. This diskette is designed to work on a 48K Apple II Plus system with a Disk II drive. A 48K Apple II system with Integer BASIC can be used if either an Applesoft Firmware or a Language System card is installed in peripheral slot 0.

It is recommended that the reader use this diskette in conjunction with the book in order to study the programs. It is further recommended that a second diskette be used to store the programs you develop from the "Posers and Problems." The following sections will explain how to boot up the Apple, initialize your own diskette, care for diskettes, use a printer, and what to do if you get into trouble.

A.2.1 Booting Up

Using the diskette labeled "An APPLE for the Teacher: Fundamentals of Instructional Computing," boot up the system as follows:

1. Open the door on disk drive 1 by pulling outward on the bottom edge of the door.
2. Slip the diskette into the slot in the front of the drive with the diskette label facing upwards. The edge of the diskette with the oval cutout should be toward the back of the drive.
3. Push the diskette gently into the drive until it is entirely inside it. Do not force or bend the diskette. Close the disk drive door.
4. Turn on the television and turn the sound down all the way.
5. Turn on the Apple by pushing upward on the switch located at the back of the computer on your left-hand side. The red light on the disk drive will go on and the drive will make clicking sounds.

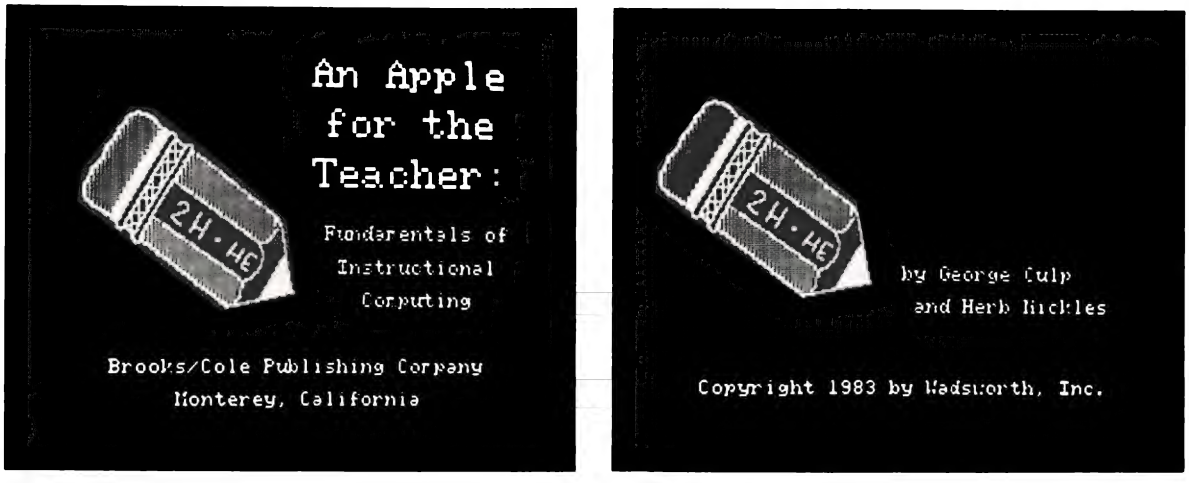


Figure A.2
(Photograph by Carey Van Loon)

6. After a few seconds, the title of this book should appear on the screen (Figure A.2), followed by the authors' names. After a few more seconds, a warning about use of the diskette and a menu of the programs stored on it will appear (Figure A.3).

7. Select a program from the menu, type in its corresponding number, and depress the RETURN key. The program then may be either LOADed or RUN at your option by depressing 1 or 2 followed by depressing the RETURN key.

The process of powering up the Apple is called *booting DOS* by experienced Apple users. What takes place is that the DOS (disk operating system) is loaded from the diskette into RAM memory and a predetermined program is executed.

To execute another program on the diskette, type RUN followed by the name of the program, and depress the RETURN key. To see a list of the program's statements, type LIST and depress the RETURN key. For example:

```
RUN PROGRAM 1        [don't forget the RETURN key]
```

will load PROGRAM 1 from the diskette into the computer's memory and execute it; and

```
LIST        [depress RETURN]
```

will list all the statements of PROGRAM 1.

Figure A.3

```
        ------------------------
        W   A   R   N   I   N   G
        ------------------------

THIS IS NOT A DEMONSTRATION DISKETTE!

THE PROGRAMS ARE AN INTEGRAL PART OF

AND SOLELY FOR USE IN CONJUNCTION WITH

THE ACCOMPANYING TEXT MATERIAL.

          DEPRESS ANY KEY...
```

[Clear screen]

```
* * M E N U  OF  P R O G R A M S * *

EXAMPLE PROGRAMS FROM THE TEXT:

1  6   11  16  21  26
2  7   12  17  22
3  8   13  18  23
4  9   14  19  24
5  10  15  20  25

ANSWERS TO 'POSERS AND PROBLEMS':

27..A354  30..A662  33..A786
28..A422  31..A784
29..A458  32..A785

DEMONSTRATION PROGRAMS FROM THE TEXT:

34..ISLAND         36..SOCKS
35..KEYWORD DEMO   37..MENU

PLEASE ENTER THE NUMBER OF THE PROGRAM
YOU WISH?1
```

[Clear screen]

```
DO YOU WISH TO:
    1. LOAD
    2. RUN

PROGRAM 1 (ENTER 1 OR 2)?1
```

[Clear screen]

```
LOADING PROGRAM 1...
```

A.2.2 Initializing a Blank Diskette

You will want to store the programs you write on a diskette. Although you can store your programs on the diskette that comes with this book, it is best to use another diskette so that you don't accidentally delete a sample program.

Obtain a new blank diskette and follow this procedure:

1. After removing the sample program diskette from the disk drive, insert your blank diskette into the disk drive.
2. Type NEW and depress the RETURN key.
3. Type 10 HOME and depress the RETURN key.
4. Type INIT HELLO and depress the RETURN key. The red light on the disk drive will glow and the drive will whirr for about two minutes.
5. When the "]" character appears, remove the diskette and label the outside of the diskette with a pressure-sensitive label. Use a felt pen so that you won't damage the diskette.

It is very important that you have a blank diskette in the drive when you follow the above procedure, otherwise you will destroy any programs on the diskette. This procedure *formats* the diskette so that it can be used with the Apple. The DOS is copied from memory onto the diskette along with whatever program is stored in memory. The diskette can subsequently be used to power up (boot) the system.

A.2.3 Care and Treatment of Diskettes

The programs you store on diskette are valuable. You have an investment in them—either time or money or both. Eliminate troubles by following these simple precautions:

1. Handle a diskette by the jacket (plastic cover) *only*. Do not allow *anything* to touch the exposed area of the diskette.
2. Never subject a diskette to a magnetic field; it may erase the diskette. Setting your diskette on top of a TV or printer could cause problems.
3. Keep diskettes flat. Do not fold, bend, or crimp in a three-ring binder.
4. Insert diskettes carefully into the disk drive. Don't use unnecessary force.
5. Store diskettes in their envelope away from liquids, dirty or greasy surfaces, and dust. In the classroom, chalk dust can cause serious problems with diskettes.
6. Do not expose diskettes to extreme hot or cold temperatures. Car dashboards and trunks are diskette killers.

A.2.4 How to Use a Printer

Since several different printers may be used with the Apple II computer, the following instructions for using a printer are generalized. Should these instructions not work, refer to the printer manual.

1. Locate the on/off switch on the printer and turn it on.
2. Check for a switch labeled *online/offline* and set for online.
3. Type PR#1 and depress the RETURN key. From now on, any text that appears on the television screen should also appear on the paper in the printer.
4. When a "]" appears, printing may be halted by typing PR#0 and depressing the RETURN key. Locate a switch on the printer labeled *linefeed* or *formfeed*. Use this switch to eject the paper so that the printout can be removed from the printer. (*Note:* The printer may need to be offline to eject the paper.)

The above instructions require that the printer interface be plugged into peripheral slot 1 inside the Apple. This is its normal location.

The default print line length is 40 characters, the same as the Apple's screen line length. Some printers can print 80 characters per line. To print 80 characters, type the following sequence of keys:

1. Type PR#1 and depress RETURN.
2. Type I while holding down the CTRL key.
3. Type 80 and depress RETURN.

A.3 WHAT TO DO WHEN ALL ELSE FAILS

A.3.1 Booting DOS Manually

Because of the number of possible configurations of Apple systems, the above instructions will not always boot the system. If you follow the instructions in Section A.2.1 and the disk light does not go on, you can manually boot the DOS as follows:

1. If a "]" or ">" appears on the screen, type PR#6 and depress the RETURN key.
2. If a "*" appears on the screen, type 6; then type P while holding down the CTRL key. Finally depress RETURN.

A.3.2 Getting Back to BASIC (Applesoft)

Through a number of different ways, it is possible to get out of Applesoft BASIC (designated by a "]" prompt) and into either Integer BASIC (designated by a ">" prompt) or the Apple monitor mode (designated by a "*" prompt). Follow these directions to return to Applesoft:

1. If a ">" appears on the screen, type FP and depress RETURN.
2. If a "*" appears on the screen, type 3D0G and depress RETURN. (That's a zero after the D.)

A.3.3 Halting a Runaway

Sometimes when you RUN a program or make a LISTing of a program you may desire to stop before it finishes. To do this, type C while holding down the CTRL key.

A.3.4 The Last Resort

If all attempts to get yourself out of the jam you're in have failed, try depressing the RESET key and following the instructions above for getting back into Applesoft. Note that depressing the RESET key during a program RUN can have disastrous results. (Some systems require the CTRL key to be held down while depressing RESET.)

The ultimate correction for problems is to turn the power off and then boot up the Apple again. This will definitely erase the program in memory, but it will not affect the diskette as long as the red light on the disk drive is not lit when you turn off the power.

If you cannot get the companion diskette to this book to boot correctly, reread Section A.2 to make sure the Apple you are using is configured correctly.

Appendix B

Applesoft Language Summary

This appendix defines the most common statements and commands used by educators on the Apple computer. It is not a complete listing of all possible statements, nor does it present a detailed description of the action of each statement. The reader who requires such information is referred to the Applesoft BASIC programming reference manual that comes with each Apple II.

The assumption of this appendix is the same as that of the rest of the text: The statements and commands as described are intended to be used on an Apple II Plus (or Apple II with an Applesoft Firmware or Language System card) with 48K of RAM memory and one or two disk drives whose controller card is located in slot #6. This configuration is very common for educational users. If the reader's system is not configured in this fashion, some of the following statements and commands will function differently than documented.

In the following summary, the general format for each statement or command is followed by an example (or examples) and a description of the action initiated. The conventions and abbreviations used are as follows:

`<...>`	Required element.
`{...}`	Optional element.
`cond`	Any logical condition.
`dimension(s)`	The maximum dimension(s) of an array.
`expr`	Any numeric constant, variable, or expression.
`file`	Any legal filename (only the first 30 characters are used).
`key`	Any key on the Apple keyboard.
`line number`	Any legal line number from 0 to 32767.
`message`	Any combination of characters.
`statement`	Any legal Applesoft statement.

`string`	Any string constant, variable, or expression.
`variable` or `var`	Any legal variable as described in Section B.4.
`X`	Any numeric constant, variable, or expression defining an x-axis value.
`Y`	Any numeric constant, variable, or expression defining a y-axis value.

B.1 BASIC STATEMENTS

DATA line number DATA <list of variables>

```
210 DATA 4.3,"A TO Z",10
```

Provides a program with data which can be stored into variables using the READ statement. In the example, 4.3 is a real number, "A TO Z" is a string, and 10 is an integer. (See READ below.)

DIM line number DIM <variable(dimension(s))>

```
10 DIM A(23),B(3,4),C$(4),D$(12,30)
```

Defines a variable capable of storing a list (single dimension) or a table (double dimension) of a specified length. In the example, A is a numeric variable with 23 possible entries. D$ is a string variable with a maximum of 12 rows and 30 columns.

END line number END

```
32767 END
```

Terminates the execution of a program.

FOR line number FOR <var> = <expr> TO <expr> {STEP <expr>}

```
45 FOR I = 2 TO 10 STEP 2
```

Creates a loop that executes all of the statements between a FOR and a NEXT statement a specified number of times. In the example, the loop would be executed for the values of I from 2 to 10 by 2s (i.e., 2, 4, 6, 8, and 10). (See NEXT below.)

GET line number GET <variable>

```
70 GET X$
```

Inputs a single character from the keyboard without the character being printed on the screen. Does not require the RETURN

key to be pressed. In the example, the input character is stored in the variable X$.

GOSUB

line number GOSUB <line number>

```
220 GOSUB 10000
```

Unconditionally branches program execution to a subroutine at the indicated line number. When a RETURN statement is encountered in the subroutine, execution is returned to the statement immediately following the GOSUB. The example will cause the program to branch to the subroutine beginning at line 10000. (See RETURN below.)

GOTO

line number GOTO <line number>

```
670 GOTO 10
```

Causes the execution of the program to branch to the indicated line number. In the example, program execution will branch from line 670 to line 10.

IF-THEN

line number IF <cond> THEN <statement>
line number IF <cond> THEN <line number>

```
55 IF A$ = "Y" THEN PRINT "CORRECT"
75 IF X < Z THEN 300
```

Causes the program to execute the indicated statement or branch to the indicated line number if a specified condition is true. If the condition is false, the statement or branch is not executed and the program continues with the execution of the next numbered statement following the IF-THEN. In the first example, CORRECT will be printed if A$ has the string value "Y". The second example will cause a branch to line 300 if the value stored in X is less than the value stored in Z.

INPUT

line number INPUT {string;} <list of variables>

```
240 INPUT "WHAT IS YOUR NAME? ";NAME$
800 INPUT A,B,C
```

Inputs data from the keyboard to be stored into the respective variables listed. Optionally, INPUT can print a string on the screen before waiting for input. The RETURN key must be pressed after the user has entered data. In the first example, the string WHAT IS YOUR NAME? will be printed on the screen, followed by the cursor. The string the user enters will be stored in NAME$. The second example will input from the keyboard three numeric values separated by commas and store them into A, B, and C, respectively.

LET

line number LET <variable> = <expr>
line number <variable> = <expr>

```
110 LET C = 100
120 P$ = "GREAT!"
130 A = 1/2 * B + H
```

Assigns the value of <expr> to <variable>. The word LET is optional. In the examples, the value 100 is stored in the variable C, the string GREAT! is stored in the variable P$, and variable A will have the value of one-half the value of B plus the value of H.

NEXT

line number NEXT <variable>

```
80 NEXT I
```

Terminates a loop begun by a FOR statement. The variable must be the same used in the corresponding FOR statement. In the example, line 80 will terminate the preceding statement: 45 FOR I = 2 TO 10 STEP 2. (See FOR above.)

ON-GOSUB

line number ON <expr> GOSUB <list of line numbers>

```
30 ON X GOSUB 10000,15000
```

Branches to the subroutine at the line numbers indicated, based on the arithmetic value of an expression. In the example, the program will branch to the subroutine at line 10000 if X is 1 and to the subroutine at 15000 if X is 2. If X is less than 1 or greater than 2, the statement immediately following the ON-GOSUB will be executed.

ON-GOTO

line number ON <expr> GOTO <list of line numbers>

```
40 ON X - Y GOTO 500,600,700
```

Branches to the line numbers indicated, based on the arithmetic value of an expression. In the example, the program will branch to line 500 if X − Y has the value 1, line 600 if X − Y has the value 2, and line 700 if X − Y has the value 3. If X − Y is less than 1 or greater than 3, then the statement immediately following the ON-GOTO will be executed.

PRINT

line number PRINT <list of variables>

```
890 PRINT "YOU GOT ";N;" QUESTIONS CORRECT"
```

Causes the computer to advance the cursor to the next line on the screen and print the values of the specified variables or strings. If in the example N had the value 9, YOU GOT 9 QUESTIONS

CORRECT would appear on the screen. See Section B.3, "Text Formatting Statements," for more information.

READ line number READ <list of variables>

```
465 READ X,Y,Z
```

Used in conjunction with the DATA statement to store data into variables within a program. When a READ statement is executed, the program will set the variables listed to the next successive values in the program's DATA statements. The example will take the next three values from the DATA statements and store them in X, Y, and Z, respectively. (See DATA above.)

REM line number REM <message>

```
10 REM PROGRAM BY IMA TEACHER
```

Inserts a REMark into the program. The message only appears when the program is LISTed; the computer ignores all REMarks when the program is RUN.

RESTORE line number RESTORE

```
360 RESTORE
```

Returns the DATA list pointer to the first value of the first DATA statement, allowing the DATA to be reread.

RETURN line number RETURN

```
10450 RETURN
```

Terminates a subroutine and returns execution to the next numbered statement following the GOSUB which called the subroutine. (See GOSUB above.)

B.2 GRAPHICS STATEMENTS

COLOR line number COLOR = <expr>

```
340 COLOR = 7
```

Sets the color to be plotted in low-resolution graphics. The <expr> is an integer between 0 and 15 that represents the following colors:

0	black	4	dark green	8	brown	12	green
1	magenta	5	grey	9	orange	13	yellow
2	dark blue	6	medium blue	10	grey	14	aqua
3	purple	7	light blue	11	pink	15	white

GR

line number GR

```
800 GR
```

Switches the display on the screen to low-resolution graphics (40 × 40 points) with four lines of text at the bottom. Clears the graphics screen to black and sets COLOR = 0 (black).

HCOLOR

line number HCOLOR = <expr>

```
460 HCOLOR = 1
```

Sets the color to be plotted in high-resolution graphics. The <expr> is an integer between 0 and 7 that represents the following colors:

0	black	2	violet	4	black	6	blue
1	green	3	white	5	orange	7	white

HGR

line number HGR

```
390 HGR
```

Switches the display on the screen to high-resolution graphics (280 × 160 points) with four lines of text at the bottom. Clears the graphics screen to black but does not change the value of HCOLOR.

HLIN

line number HLIN <X1>,<X2> AT <Y>

```
1010 HLIN 5,25 AT 20
```

Draws a horizontal line on the low-resolution graphics screen at the y-axis position <Y> from the x-axis position <X1> to the x-axis position <X2>. The color will be that most recently set by the COLOR statement. In the example, a horizontal line will be drawn from X = 5 to X = 25 at Y = 20.

HPLOT

line number HPLOT <X>,<Y>
line number HPLOT <X1>,<Y1> TO <X2>,<Y2>
line number HPLOT TO <X>,<Y>

```
200 HPLOT 100,130
210 HPLOT 0,0 TO 279,159
220 HPLOT TO 150,10
```

Plots dots or lines on the high-resolution graphics screen using the color most recently set by the HCOLOR statement. The high-resolution screen uses an (X,Y) coordinate system with 0,0 in the upper left corner. In the first example, a dot will be plotted at X = 100, Y = 130. In the second example, a line will be plotted from X = 0, Y = 0 (upper left corner) to X = 279, Y = 159 (lower right corner). In the third example, a line will be plotted from the last point plotted to X = 150, Y = 10.

PLOT line number PLOT <X>,<Y>

```
275 PLOT 20,30
```

Plots rectangular blocks on the low-resolution graphics screen using the color most recently set by the COLOR statement. The low-resolution screen uses an (X,Y) coordinate system with 0,0 in the upper left corner and 39,39 in the lower right corner. The example will plot a block at X = 20, Y = 30.

SCRN line number <var> = SCRN(<X>,<Y>)

```
620 Z = SCRN(27,5)
```

SCRN is the low-resolution graphics screen function that returns the color value of the graphic coordinates specified. In the example, Z will be set to the value of the color at X = 27, Y = 5.

TEXT line number TEXT

```
990 TEXT
```

Sets the screen to the text mode of 24 lines of text with 40 characters per line. TEXT does not clear the screen or HOME the cursor.

VLIN line number VLIN <Y1>,<Y2> AT <X>

```
730 VLIN 0,39 AT 20
```

Draws a vertical line on the low-resolution graphics screen at the x-axis position <X> from the y-axis position <Y1> to the y-axis position <Y2>. The color will be that most recently set by the COLOR statement. In the example, a vertical line will be drawn from Y = 0 to Y = 39 at X = 20.

B.3 TEXT FORMATTING STATEMENTS

COMMA (,) line number PRINT <var>,<var>

```
370 PRINT QUANTITY,PRICE,TOTAL
```

Used in a PRINT statement to space data into 16-column fields. In the example, the value of the variable QUANTITY will be printed in column 1, the value of the variable PRICE will be printed in column 17, and the value of the variable TOTAL will be printed in column 33.

FLASH line number FLASH

```
1500 FLASH
```

Sets the text printing mode to flashing characters. All text printed after this statement will flash. NORMAL reverses this action.

HOME line number HOME

```
10 HOME
```

Clears the text screen and returns the cursor to the home position in the upper left corner.

HTAB line number HTAB <expr>

```
550 HTAB 27
```

Moves the cursor to the specified column number (1 to 40). The HTAB statement is usually followed by a PRINT statement. In the example, the cursor will be moved to column 27.

INVERSE line number INVERSE

```
345 INVERSE
```

Sets the text printing mode to black-on-white characters instead of white on black. All text printed after this statement will be printed in inverse. NORMAL reverses this action.

NORMAL line number NORMAL

```
610 NORMAL
```

Sets the text printing mode to normal white-on-black characters. Reverses the action of the FLASH and INVERSE statements.

POS line number <var> = POS(<expr>)

```
730 X = POS(0)
```

POS is the text function that returns the current horizontal cursor position (0 to 39). Although <expr> is required, the expression has no effect on the results. In the example, X will be set to the current horizontal cursor position.

SEMICOLON (;) line number PRINT <string>;<var>

```
840 PRINT "YOU GOT ";N;" CORRECT."
```

Used in a PRINT statement to position the cursor immediately after the string or variable preceeding the semicolon. If N = 10 in the example, the printed line would read:

```
YOU GOT 10 CORRECT.
```

SPC line number PRINT <var>;SPC(<expr>);<var>

```
480 PRINT A;SPC(10);B
```

Used in a PRINT statement to insert a specified number of spaces between two variables when preceded and followed by semicolons. In the example, the value of A will be printed, followed by 10 spaces and then the value of B.

SPEED line number SPEED = <expr>

```
160 SPEED = 200
```

Sets the speed at which characters are printed on the screen. The default speed, 255, is the fastest system speed. Zero is the slowest speed.

TAB line number PRINT TAB(<expr>);<var>

```
80 PRINT TAB(25);R
```

Used in a PRINT statement to move the cursor to the specified column, where 1 is the left margin and 40 is the right margin. TAB can only move the cursor to the right. Use HTAB to move the cursor to the left. In the example, the value of R will be printed starting in column 25.

VTAB line number VTAB <expr>

```
120 VTAB 18
```

Moves the cursor to the specified line number. The top of the screen is line 1; while the bottom is line 24. The VTAB statement is usually followed by a PRINT statement. In the example, the cursor will be moved to line 18.

B.4 SUMMARY OF VARIABLE TYPES

INTEGER Variable name: Single letter (optionally followed by a single letter or digit) followed by the "%" character.

Range: −32767 to +32767.

Examples: I%, B2%, GH%

REAL Variable name: Single letter (optionally followed by a single letter or digit.)

Range: −9.99999999 E+37 to +9.99999999 E+37

Examples: S, R5, DE

STRING Variable name: Single letter (optionally followed by a single letter or digit) followed by the "$" character.

Range: 0 to 255 characters

Examples: F$, K9$, XY$.

Note that variable names may be longer than two characters, but only the first two characters are significant. Consequently, APPLE and APPLIANCE are the same real variable, AP.

B.5 SUMMARY OF OPERATORS

ARITHMETIC

+	addition
/	division
^	exponentiation (raise to a power)
*	multiplication
−	subtraction or negation

LOGICAL

AND	logical product
NOT	logical negation
OR	logical sum

RELATIONAL

=	equals
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<>	not equal to

STRING + concatenation

B.6 MATHEMATICAL FUNCTIONS

ABS

line number <var> = ABS(<expr>)

```
100 X = ABS(-6.75)
```

Returns the absolute value of <expr>. In the example, X = 6.75.

ATN

line number <var> = ATN(<expr>)

```
100 X = ATN(1)
```

Returns the arctangent of <expr> in radians. In the example, X = .785398163.

COS

line number <var> = COS(<expr>)

```
100 X = COS(1)
```

Returns the cosine of <expr>. <expr> must be in radians. In the example, X = .540302306.

EXP

line number <var> EXP(<expr>)

```
100 X = EXP(1)
```

Returns the value e∧<expr>, where e = 2.7182828183. In the example, X = 2.71828183.

INT

line number <var> = INT (<expr>)

```
100 X = INT (4.53)
```

Returns the greatest integer in <expr> which is less than or equal to <expr>. In the example, X = 4.

LOG

line number <var> = LOG(<expr>)

```
100 X = LOG(2)
```

Returns the natural logarithm of <expr>. In the example, X = .693147181.

RND

line number <var> = RND(<expr>)

```
100 X = RND(1)
```

Returns a random number greater than or equal to 0 and less than 1. If <expr> is positive, a unique set of random numbers is generated. If <expr> is 0, then the last random number gen-

erated is returned. If <expr> is negative, the same set of random numbers will be generated every time the program is run.

SGN line number <var> = SGN(<expr>)

```
100 X = SGN(-217.456)
```

Returns the sign of <expr>: +1 if positive, 0 if zero, and −1 if negative. In the example, X = −1.

SIN line number <var> = SIN(<expr>)

```
100 X = SIN(1)
```

Returns the sine of <expr>. <expr> must be in radians. In the example, X = .841470985.

SQR line number <var> = SQR(<expr>)

```
100 X = SQR(16)
```

Returns the square root of <expr>. In the example, X = 4.

TAN line number <var> = TAN(<expr>)

```
100 X = TAN(1)
```

Returns the tangent of <expr>. <expr> must be in radians. In the example, X = 1.55740772.

B.7 STRING FUNCTIONS

ASC line number <var> = ASC(<string>)

```
100 X = ASC("APPLE")
```

Returns the ASCII code for the first character in the string specified. In the example, X = 65.

**CHR$** line number <string> = CHR$(<expr>)

```
100 X$ = CHR$(65)
```

Returns the ASCII character specified by the numerical value of <expr>. In the example, X$ = "A"

**LEFT$** line number <string> = LEFT$(<string>,<expr>)

```
100 X$ = LEFT$("APPLE",3)
```

Returns a substring of <string> from the first character to the <expr>th character. In the example, X$ = "APP".

LEN line number <var> = LEN(<string>)

```
100 X = LEN("APPLE")
```

Returns the number of characters contained in <string>. In the example, X = 5.

**MID$** line number <string> = MID$(<string>,<expr1>,<expr2>)

```
100 X$ = MID$("NOW IS THE TIME",5,6)
```

Returns the substring of <string> that begins with the character specified by <expr1> and has a length of <expr2> characters. In the example, X$ = "IS THE".

**RIGHT$** line number <string> = RIGHT$(<string>,<expr>)

```
100 X$ = RIGHT$("APPLE",2)
```

Returns the substring of <string> consisting of the rightmost characters specified by <expr>. In the example, X$ = "LE".

**STR$** line number <string> = STR$(<expr>)

```
100 X$ = STR$(24.07)
```

Converts the <expr> to a string. In the example, X$ = "24.07".

VAL line number <var> = VAL(<string>)

```
100 X = VAL("365 DAYS")
```

Converts the <string> to a real or integer variable. The conversion will terminate when a non-numeric character is encountered. In the example, X = 365.

B.8 BASIC AND DISK COMMANDS

CATALOG CATALOG {,D<expr>}

```
CATALOG,D2
```

Prints a list of all the files on a diskette. Optionally the disk drive number, D<expr>, may be specified. In the example, a catalog of the diskette in drive two will be listed on the screen.

DEL

DEL <line number>,<line number>

```
DEL 350,400
```

Deletes line numbers from the program in memory starting with the first line number specified and ending with the second line number specified. In the example, line 350, line 400, and all of the lines with numbers between 350 and 400 will be deleted.

DELETE

DELETE <file> {,D<expr>} {,V<expr>}

```
DELETE BUTTERFLIES
```

Erases a file from a diskette. Optionally, the drive number or volume number may be specified. In the example, the file BUTTERFLIES will be erased from the diskette in the drive last used.

INIT

INIT <file> {,D<expr>} {,V<expr>}

```
INIT HELLO, V25
```

Initializes a blank diskette so that it can be used. The current program in memory will be saved as the <file> specified, and that program will be run when the diskette is booted. Optionally, the drive number or volume number may be specified. In the example, the diskette in the drive most recently used will be initialized as volume 25 with the program in memory stored as HELLO.

LIST

LIST {<line number>} {,<line number>}

```
LIST
LIST 300
LIST 1000,2000
```

Lists lines of the program in memory on the screen. Optionally, a line number or range of line numbers may be specified. In the first example, the entire program will be listed. In the second example, line 300 only will be listed. In the third example, lines 1000 to 2000, inclusive, will be listed.

LOAD

LOAD <file> {,D<expr>} {,V<expr>}

```
LOAD SNOW WHITE
```

Loads the specified file from a diskette into memory. The current program in memory will be erased. Optionally, the drive number or volume number may be specified. In the example, the program SNOW WHITE will be loaded into memory from the disk most recently used.

LOCK

LOCK <file> {,D<expr>} {,V<expr>}

```
LOCK MATH DRILL
```

Protects a file from being replaced or deleted accidentally. The UNLOCK command will reverse the action. Optionally, the drive number or volume number may be specified. In the example, the file MATH DRILL will be LOCKed on the diskette in the drive most recently used.

NEW

```
NEW
```

Erases the program and variables currently in memory. Used to clear memory before writing a new program.

PR

PR#<expr>

```
PR#6
PR#1
PR#0
```

Transfers output to the specified peripheral slot number. In the examples, PR#6 boots disk drive 1; PR#1 transfers all subsequent output to a printer, assuming the printer interface is in slot 1; PR#0 returns output to the screen.

RUN

RUN {<file>} {,D<expr>} {,V<expr>}

```
RUN SPELL
```

Executes the program in memory if no file is specified. If a file is specified, memory is cleared, the file is loaded from a diskette, and the program is executed. Optionally, the drive number or volume number may be specified. In the example, the program SPELL will be loaded from the most recently used disk drive and executed.

RENAME

RENAME <file1>,<file2> {,D<expr>} {,V<expr>}

```
RENAME PROGRAM 1,MUSCLES
```

Changes the name of <file1> to <file2> on a diskette. Optionally, the drive number or volume number may be specified. In the example, PROGRAM 1 will be renamed MUSCLES on the diskette in the drive most recently used.

SAVE

SAVE <file> {,D<expr>} {,V<expr>}

```
SAVE PICKLES
```

Saves the program currently in memory on diskette as the file specified. Optionally, the drive number or volume number may be specified. If the file specified already exists on the diskette, it will be replaced by the program in memory unless it was LOCKed. In the example, the program in memory will be saved with the name PICKLES on the diskette in the drive most recently used.

UNLOCK

UNLOCK <file> {,D<expr>} {,V<expr>}

```
UNLOCK MATH DRILL
```

Removes the accidental replace or delete lock on the file specified. Optionally, the drive number or volume number may be specified. In the example, the file MATH DRILL will be unlocked on the diskette in the drive most recently used.

B.9 SPECIAL KEYS

ARROW KEYS

LEFT ARROW (←)
RIGHT ARROW (→)

The two keys on the Apple keyboard marked with a left arrow and right arrow are used to edit programs. The LEFT ARROW is used to delete characters previously typed in the current line. The RIGHT ARROW will reenter a character on the screen as though you were typing it.

CONTROL (CTRL)

CONTROL <key>

CONTROL C
CONTROL G
CONTROL X

The CONTROL key is used in conjunction with other keys to specify a variety of actions. To execute a CONTROL sequence, hold the CONTROL key down and then depress the other key. In the examples, CONTROL C will break the execution of a program and print the line number at which execution terminated, CONTROL G will sound a bell on the Apple speaker, and CONTROL X will delete the current line being typed.

ESCAPE (ESC)

ESCAPE <key>

ESCAPE I
ESCAPE J
ESCAPE K
ESCAPE M

The ESCAPE key is commonly used to edit programs. When the ESCAPE key is typed, the moveable-cursor mode is entered. The keys I, J, K, and M are used to move the cursor up, left, right, and down, respectively. Once the cursor is positioned, any key except I, J, K, and M will return to normal mode. The LEFT ARROW and RIGHT ARROW keys may then be used to make edits.

REPEAT (REPT) REPEAT <key>

When the REPEAT key is held down in conjunction with another key, the other key will be repeatedly typed.

RESET RESET
CONTROL RESET

The RESET key immediately halts the execution of a program and sets the screen to TEXT mode. If RESET is typed while a program is being saved on a diskette, the file may be damaged. For this reason, newer Apples have an internal switch which can be set to require the CONTROL key to be held down while typing RESET.

B.10 ASCII CHARACTER CODES

The following codes are used in the CHR$ and ASC functions:

Code	Character	Code	Character
0	CTRL @	48	0
1	CTRL A	49	1
2	CTRL B	50	2
3	CTRL C	51	3
4	CTRL D	52	4
5	CTRL E	53	5
6	CTRL F	54	6
7	CTRL G (bell)	55	7
8	CTRL H (←)	56	8
9	CTRL I	57	9
10	CTRL J	58	:
11	CTRL K	59	;
12	CTRL L (form feed)	60	<
13	CTRL M (return)	61	=
14	CTRL N	62	>
15	CTRL O	63	?

Code	Character	Code	Character
16	CTRL P	64	@
17	CTRL Q	65	A
18	CTRL R	66	B
19	CTRL S	67	C
20	CTRL T	68	D
21	CTRL U (→)	69	E
22	CTRL V	70	F
23	CTRL W	71	G
24	CTRL X	72	H
25	CTRL Y	73	I
26	CTRL Z	74	J
27	ESC	75	K
28	not available	76	L
29	CTRL SHIFT M	77	M
30	CTRL ∧	78	N
31	not available	79	O
32	SPACE	80	P
33	!	81	Q
34	"	82	R
35	#	83	S
36	$	84	T
37	%	85	U
38	&	86	V
39	'	87	W
40	(	88	X
41	)	89	Y
42	*	90	Z
43	+	91	[
44	,	92	not available
45	−	93	]
46	.	94	∧
47	/	95	not available

Appendix C

Answers to Selected Questions and Problems

CHAPTER 1

Think About This (for Fun)

One Word

Text Questions

Section 1.5.3 The output would be "close packed" (printed with no separating spaces).

Section 1.5.5 The blank space is needed to separate the comma from the name (value of N$). Otherwise, the comma and name would be close packed, as in HOWDY,SAMMY.

Posers and Problems

1.
```
10 PRINT "HELLO"
20 PRINT "WHAT'S YOUR HEIGHT IN INCHES";
30 INPUT H
40 M = 2.54 * H
50 PRINT "YOU ARE ";M;" CENTIMETERS TALL!"
60 END
```

2. 25, 4, 6, .66666667, 3

3. So that any output will be close packed. Commas cause the output to be tabbed 16 spaces before being printed.

4. The semicolon close packed the "?" printed by execution of the INPUT statement (60).

5. See Section 2.4 of Chapter 2.

6.
```
NAME          SCORE         AVERAGE
←16 spaces→←16 spaces→
```

7. Enter and RUN the program.

8.
```
10 PRINT "DEGREES CELSIUS";
20 INPUT C
30 F = (C * 9/5) + 32
40 PRINT C;" DEGREES C = ";F;" DEGREES F."
50 END
```

9.
```
10 PRINT "HOW MANY CUPS";
20 INPUT C
30 PRINT "HOW MANY OUNCES";
40 INPUT Z
50 T = (8 * C) + Z
60 PRINT C; " CUPS ";Z;" OUNCES = ";T;"
   TOTAL OUNCES."
70 END
```

10.
```
10 PRINT "FIRST NAME";
20 INPUT F$
30 PRINT "LAST NAME";
40 INPUT L$
50 PRINT "HELLO, ";F$;" ";L$;"!"
60 END
```

CHAPTER 2

Think About This (for Fun)

A chair, a bed, and a toothbrush.

Text Questions

Section 2.3 INT(10 * .99999999 + 1) = 10
INT(10 * .01 + 1) = 1
Range of INT(10 * RND(1) + 3) = 12 to 3
Range of INT(9901 * RND(1) + 100)/100 = 100.00 to 1.00
Range of INT(91 * RND(1) + 5) = 95 to 5

Posers and Problems

1. R is for *numeric* input (years)
   R$ is for *string* input (state name)

2. There is no real difference. Random output would still be given. If X is 1, then 440 is PRINTED, and so on.

3. Change the text of PRINT statements 400, 420, and 440.

4. Change PRINT statements 200 and 210 to ask for your age; change the "39" in statements 230–250 to your age.

5.
```
380 X = INT(5 * RND(1) +1)
390 ON X GOTO 400,420,440,452,456
452 PRINT "HO-HO! NOT THAT OLD!"
454 GOTO 140
456 PRINT "COME DOWN SOME!"
458 GOTO 140
```

6.
```
70 PRINT "WHAT STATE IS THE THIRD"
80 PRINT "LARGEST BY LAND AREA";
100 IF R$ = "CALIFORNIA" THEN 130
110 PRINT "NO, IT'S CALIFORNIA!"
130 PRINT "RIGHT! WHERE THE ORANGES GROW!"
```

7.
```
1 PRINT "WHAT'S YOUR FIRST NAME";
2 INPUT F$
510 PRINT,"BYE-BYE, ";F$;"!"
```

9. X = INT(76 * RND(1) + 25)

10. 29 to 5, inclusive

11.
```
10 PRINT "WHAT'S YOUR HEIGHT IN INCHES";
20 INPUT H
30 IF H > 72 THEN 80
40 IF H < 60 THEN 100
50 PRINT
60 PRINT "AVERAGE"
70 GOTO 110
80 PRINT "TALL"
90 GOTO 110
100 PRINT "SHORT"
110 END
```

12.
```
10 PRINT "ENTER THREE SIDES (SEPARATED BY
   COMMAS),"
20 PRINT "WITH THE LONGEST SIDE ENTERED
   LAST"
30 INPUT A,B,C
40 IF A > C THEN 90
50 IF B > C THEN 90
60 IF (C ^ 2) = ((A ^ 2) + (B ^ 2)) THEN
   110
70 PRINT "NOT A RIGHT TRIANGLE"
80 GOTO 120
90 PRINT "LONGEST SIDE NOT ENTERED LAST!"
100 GOTO 10
110 PRINT "THAT'S A RIGHT TRIANGLE!"
120 END
```

13.
```
10 PRINT "ENTER ANY NUMBER, 1-10,
   INCLUSIVE";
20 INPUT N
30 A$ = "WAS ENTERED."
40 IF N = 3 THEN 90
50 IF N = 6 THEN 110
60 IF N = 9 THEN 130
70 PRINT "NEITHER 3, 6, OR 9 ";A$
80 GOTO 140
90 PRINT "THREE ";A$
100 GOTO 140
110 PRINT "SIX ";A$
120 GOTO 140
130 PRINT "NINE ";A$
140 END
```

CHAPTER 3

Think About This (for Fun)

In

Text Questions

Section 3.2.2 A new value (2) was READ and assigned to N at statement 60.

ABC would be printed.

A new value (DEF) was READ and assigned to N$ at statement 40.

Use of commas (tabs) and semicolons (close packs).

Section 3.2.3 After statement 30 is executed, the data pointer is "past" the last data element.

The error was caused by no DATA present to be READ (the data pointer is "past" the last element).

Statement 60 increases X by 1 each time it is executed. When X is equal to 2 (statement 50), transfer is to statement 100 (END).

The program would endlessly READ, PRINT, and RESTORE.

Posers and Problems

1.
```
10 FOR Y = 1 TO 10

10 DATA 1,"ABC","DEF"
(or)
20 READ N,N$,N1
30 PRINT N,N$,N1
```

```
DEL 60,70
(or)
55 RESTORE
(or)
10 DATA 4,5,6,7
```

2.
```
40 PRINT S$,S
50 NEXT I
60 END
```

3.
```
5 T = 0
42 T = T + S
44 X = X + 1
55 PRINT "THE AVERAGE SCORE IS ";T/X;"!"
```

4. LOAD, RUN, and LIST program A354 from the text diskette.

5. FOR X = 10 TO 1 STEP -1 starts at 10 and "counts" the loop to 1 in increments of −1.

   The comma in statement 55 tabs 16 spaces before printing.

   The ";" in statement 70 close packs the "tails" (*).

   The 90 PRINT statement "cancels" the close packing of the semicolon in statement 70.

6.
```
10 PRINT "CELSIUS","FAHRENHEIT"
20 PRINT "-------","----------"
30 FOR C = 0 TO 100 STEP 5
40 F = 32 + (C * 9/5)
50 PRINT C,F
60 NEXT C
70 END
```

7.
```
10 FOR I = 1 TO 10
20 PRINT I;" CUBED IS ";I ^ 3
30 NEXT I
40 END
```

CHAPTER 4

Think About This (for Fun)

A 50-cent piece and a nickel (one of the coins is not a nickel—although the other one is!)

Text Questions

Section 4.2.1 N$(3) = PHIL; S(4) = 35; the two lists would be printed.

The lists would be printed, but in reverse order (4 to 1).

Section 4.2.2 S(1,1) = 95; S(3,2) = 93

For the complete program in Section 4.2.2, LOAD, RUN, and LIST program A422 on the text diskette.

Section 4.2.2 (cont.) The comma in statements 65 and 81 are needed to tab before printing the first and second scores. The PRINT in 95 is needed to cancel the tab effect of statement 81.

Posers and Problems

1.
```
10 REM N$()=NAME; S()=SEM.AVE.; F()=FINAL
   EXAM
20 DIM N$(20),S(20),F(20)
30 FOR I = 1 TO 20
40 READ N$(I),S(I),F(I)
50 PRINT N$(I),S(I),F(I)
60 NEXT I
.
.
.
1000 REM DATA FOR 20 STUDENTS AND THEIR
     SCORES
1010 DATA "JONES",80,82, [etc.]
.
.
.
2000 END
```

2.
```
10 REM N$()=NAME, S( , )=STUDENT SCORES
20 DIM N$(25),S(25,3)
30 FOR I = 1 TO 25
40 READ N$(I)
50 FOR J = 1 TO 3
60 READ S(I,J)
70 NEXT J
80 NEXT I
.
.
.
1000 REM DATA FOR 25 STUDENTS, EACH WITH
     3 SCORES
1010 DATA "ABEL",95,80,88, *etc.+
```

4. Three states would be randomly selected (with a random chance that one would be repeated).

5. Three states would be randomly selected without any state being repeated.

6. Five states would be printed without any repetition. Then the program becomes an endless loop, trying to find the sixth state not yet printed (FOR K = 1 TO 6) when only five states were given.

7.
```
5 F = 0
25 H$ = "STEPHEN F. --?--"
55 H$ = "DICK AND JANE'S DOG."
212 IF F = 1 THEN 220
214 F = 1
216 PRINT "HINT: ";H$
218 GOTO 200
250 F = 0
252 RETURN
```

8. LOAD, RUN, and LIST program A458 on the text diskette.

9.
```
10 DIM Z(10)
20 FOR I = 1 TO 4
30 X = INT(10 * RND(1) + 1)
40 IF Z(X) = 1 THEN 30
50 Z(X) = 1
60 PRINT X
70 NEXT I
80 END
```

CHAPTER 5

Think About This (for Fun)

The man opened one carton, took one package, opened it, and then dropped one cigarette overboard. This made the raft a cigarette lighter!

Posers and Problems

1.
```
'
'
'
200 INPUT R$
210 IF R$ = A$ THEN 260
220 IF F = 1 THEN 300
230 F = 1
240 PRINT H$ [Give a hint]
250 GOTO 200
260 IF F = 1 THEN 280 [Skip giving credit]
270 C = C + 1 [Give credit]
280 PRINT "VERY GOOD!"
290 GOTO 310
300 PRINT "A CORRECT ANSWER IS ";A$
310 RETURN
'
'
'
```

2. See statements 220–250 in the above program fragment.

3.
```
10 REM Q$()=QUESTION; A$()=ANSWER;
   Z()=FLAG
20 DIM Q$(10),A$(10),Z(10)
30 FOR I = 1 TO 10
40 READ Q$(I),A$(I)
50 NEXT I
60 FOR Q = 1 TO 5
70 X = INT(10 * RND(1) + 1)
80 IF Z(X) = 1 THEN 70
90 Z(X) = 1
100 PRINT Q$(X)
'
'
'
1000 REM DATA FOR 10 QUESTIONS, ANSWERS
1010 DATA "QUESTION 1","ANSWER 1", [etc.]
'
'
'
2000 END
```

5. LOAD, RUN, and LIST PROGRAM 16 from the text diskette. (PROGRAM 16 is an example program in Chapter 6.)

CHAPTER 6

Think About This (for Fun)

There are 6 F's (the "of's" are often overlooked).

Text Questions

Section 6.7.2 For five choices, make the following modifications:

```
270 DIM A$(5),R$(5)
5050 FOR I = 1 TO 5
5110 PRINT "YOUR CHOICE (1-5)";
5150 IF R > 5 THEN 5110
```

Then add additional data elements for each fifth choice and its response.

CHAPTER 7

Think About This (for Fun)

SLEEPLESSNESS (as in programming)

Posers and Problems

1.
```
10 HGR
20 HCOLOR = 3
30 HPLOT 10,10 TO 100,100
40 END
```

2.
```
'
'
'
440 REM PLOT SQUARE
460 LET A = INT(RND(1) * 40)
470 HLIN 0,A AT 0
480 VLIN 0,A AT 39
490 HLIN A,0 AT 39
500 VLIN A,0 AT 0
510 NEXT I
520 END
DEL 530,680
'
'
'
```

3. The screen will change from text to high-resolution graphics, and the entire screen will be colored violet.

4.
```
10 HOME
20 GR
30 FOR I = 0 TO 15
40 COLOR = I
50 VLIN 0,39 AT I * 2 + 5
60 VLIN 0,39 AT I * 2 + 6
70 NEXT I
80 END
```

5.
```
10 HOME
20 GR
30 COLOR = 6
40 FOR Y = 0 TO 39
50 HLIN 0,39 AT Y
60 NEXT Y
70 COLOR = 13
80 FOR Y = 0 TO 38 STEP 2
90 FOR X = 0 TO 38 STEP 2
100 PLOT X,Y
110 NEXT X
120 NEXT Y
130 FOR Y = 1 TO 39 STEP 2
140 FOR X = 1 TO 39 STEP 2
150 PLOT X,Y
160 NEXT X
170 NEXT Y
180 END
```

6.
```
10 HOME
20 HGR
30 HCOLOR = 3
40 FOR Y = 0 TO 159
50 X = SQR (Y) * 20
60 HPLOT X,Y
70 NEXT Y
80 END
```

or

```
10 HOME
20 HGR
30 HCOLOR = 3
35 HPLOT 0,0
40 FOR Y = 1 TO 159
50 X = SQR(Y) * 20
60 HPLOT TO X,Y
70 NEXT Y
80 END
```

CHAPTER 8

Think About This (for Fun)

50½ + 49⅜ = 100

CHAPTER 9

Think About This (for Fun)

28 jumps (after 27 jumps, the frog is 3 feet from the top of the well; one more jump of 3 feet is needed).

Appendix D

Annotated Bibliography

The periodicals listed in the following pages were selected from a more complete bibliography developed by Ron Adams of The College of New Caledonia in Mackenzie, British Columbia, Canada. These periodicals were judged by the authors as being most useful to the teacher using an Apple microcomputer in the classroom. The prices quoted are for one-year subscriptions taken out in Summer 1982. (*Note:* Any comments found in the abstracts are those of Professor Adams and not the authors.)

ADCIS NEWSLETTER U.S.A. $40 membership
Computer Center CANADA $40
Western Washington University
Bellingham, WA 98255

This newsletter is published every two months by the Association for the Development of Computer-Based Instructional Systems, one of the oldest and best-organized groups of post secondary computer-using educators. It has special-interest groups for those developing software for health education, home economics, mathematics, music, PLATO, elementary and secondary schools, computer-based training, and the handicapped. ADCIS annually hosts one of the most important computers-in-education conferences, and publishes the quarterly *Journal of Computer-Based Instruction*, which provides some of the most scholarly articles in this field.

APPLE ASSEMBLY LINE U.S.A. $12
S-C Software CANADA $12
Box 280300
Dallas, TX 75228

A monthly newsletter featuring beginners' tutorials, utility programs, and programming techniques in Apple assembly language.

APPLE EDUCATION NEWS Free
Box 20485
San Jose, CA 95106

This newsletter, published occasionally by the Apple Corporation, is a useful source of information on the burgeoning Apple-based CAI programs being developed in American universities, colleges, and schools. It also contains the inevitable glowing performance accounts of the company's educational software, as well as of the software being developed for the Apple by other companies. The newsletter is evidently intended for distribution in computer stores because there is no subscription information in it.

APPLE EDUCATORS' NEWSLETTER U.S.A. $15
9525 Lucerne Street CANADA $25
Ventura, CA 93004

The newsletter is published every two months by the Apple for the Teacher group, a California-based organization of elementary and high school teachers dedicated to sharing and developing educational software reviews and information on CAI projects that don't get coverage in the glossy magazines. Recommended.

APPLE MAGAZINE Free
10260 Bandley Drive
Cupertino, CA 95014

The Apple Corporation's quarterly catalogue disguised as a magazine. It contains several short articles lauding the Apple's versatility as a personal and small-business computer, and provides an up-to-date listing of all hardware and software marketed by the company. It's available at Apple dealers and usually is given free to potential customers.

APPLE ORCHARD U.S.A. $15
910A George Street CANADA $22.50
Santa Clara, CA 95050

The quarterly magazine of the International Apple Core, a loosely structured umbrella organization of 200 Apple-user groups around the world. It contains useful utility programs, programming tips, short articles on computer literacy, a column on new products for the Apple, and an occasional interview—most of which are reprinted from member groups' newsletters. For example, the Spring 1981 issue contains an interview with Apple founder Steve Wozniak and a tutorial on text-formatting far superior to the explanation in the Apple manuals. The optional $100 IAC club membership includes subscription to *Apple Orchard*, five diskettes of contributed software, and a newsletter of technical notes on the Apple.

APPLEGRAM CANADA $15 membership
Apples B.C. Computer Society
316–8055 Anderson Road
Richmond, B.C.
Canada V6Y 1S2

The quarterly newsletter of the Apples B.C. Computer Society, which includes several of the province's top microcomputer programmers. It contains software reviews, program listings, short articles on technical topics, buy-and-sell advertisements, club notices, and a list of programs that members contributed to a software library that can be bought for $10 a disk.

BOUNTY U.S.A. $6
17710 De Witt Avenue
Morgan Hill, CA 95037

A quarterly newsletter for special education teachers that includes a "computer corner" devoted to reviews of microcomputer software for the learning disabled.

BYTE U.S.A. $19
Box 590 CANADA $21
Martinsville, NJ 08836

Byte is McGraw-Hill's monthly attempt to imitate the worst features of *Scientific American* and the Sears catalog. The articles are not only highly technical and frequently devoted to arcane subjects that will bewilder the small-systems user, but also buried in a blizzard of advertising that occasionally pushes this unexceptional magazine to 500 pages. Each issue has a theme (local network and data base management systems have been featured in recent months) and a do-it-yourself project by electronics wizard Steve Ciarcia. Indeed, *Byte* may well be the only microcomputer periodical that will appeal to computer professionals and hobbyists who have their basements crammed with gadgetry. Nonetheless, it's the leading microcomputer journal and the advertisements will keep you abreast of the latest developments and all the new hardware in the field. I buy it for three reasons: Bytelines, which is Sol Libes's fascinating analysis of news and rumors in the microcomputing industry; the Education Forum, which reports CAI projects at various American universities and colleges; and Robert Tinney's remarkable covers, which will surely become 20th century classics.

CALL-A.P.P.L.E. U.S.A. $40 membership
304 Main Avenue South CANADA $40
Suite 300 membership
Renton, WA 98055 Subsequent years $15

Published nine times a year, *Call-A.P.P.L.E.* is the magazine of the oldest and most sophisticated of the Apple-users groups—the Apple Puget Sound Program Library Exchange. It is an excellent source of programming tips, utility program listings, software reviews, and product reports. Although it is geared to experienced users, beginners and experts alike may call a "hot line" for advice from 9 a.m. to 3 p.m. and 6 p.m. to 10 p.m. seven days a week. The hot line has saved me many hours of frustration. I strongly recommend membership in *Call-A.P.P.L.E.* for serious Apple users.

THE CATYLIST U.S.A. $12
1259 El Camino Real
Suite 275
Menlo Park, CA 94025

Published six times a year, *The Catylist* features articles and reports on microcomputers in special education.

CLASSROOM COMPUTER NEWS U.S.A. $16
Box 266 CANADA $21
Cambridge, MA 02139

Classroom Computer News is probably the best source of educational software reviews. My complimentary issue contained the following articles: "Computer literacy—What Should Schools Be Doing About It?"; "A School Administrator Looks at Visicalc"; "Special Tools for Special Needs"; "Programming 1—The Starting Gate"; "Word Processors for Teachers"; "Microcomputers in the School Library's Future"; and "How Does the Computer Remember All That Stuff?" Highly recommended for teachers.

COMPUKIDS U.S.A. $12
Box 874
Sedalia, MO 65301

A monthly newsletter for children that contains stories, games, and contests that will appeal to junior microcomputer enthusiasts.

COMPUTERTOWN USA! Donations
Box E
Menlo Park, CA 94025

The monthly newsletter of a group of enthusiasts dedicated to promoting computer literacy in Menlo Park, California. Their projects include microcomputer fairs, programming contests, and public demonstrations, as well as the installation of microcomputers in public libraries. It's a good source of ideas on grass-roots microcomputer uses. For example, the August 1981 issue has an article on raising funds for the purchase of microcomputers.

THE COMPUTING TEACHER U.S.A. $14.50
Computing Center CANADA $20.00
Eastern Oregon State College
La Grande, OR 97850

Edited by David Moursund, a leading CAI educator, this journal focuses on teacher education, computer-assisted instruction, and the impact of computers on curriculums. Apart from the usual articles and software reviews, *The Computing Teacher* features reports on CAI projects, articles on instructional design, and an assortment of calculator and microcomputer programming assignments that can be adapted to classroom use. Any teachers using a microcomputer should not be without this journal.

CREATIVE COMPUTING U.S.A. $25
Box 789–M CANADA $30
Morrison, NJ 07960

If you plan to purchase only one magazine, this should be your choice. *Creative Computing* has all the types of articles, reviews, program listings, and columns usually appearing in other microcomputer monthlies, as well as a sense of humor: It's sprinkled with cartoons, delightful pen-and-ink drawings, satirical pieces, short stories, puzzles, and even the occasional poem. Moreover, it's not restricted to computer topics. Recent issues had an excellent series on effective writing techniques that any writer can use. The judgement of its editors, however, appears shortsighted: they rejected this critical bibliography with a typewritten form letter.

CUE NEWSLETTER U.S.A. $6
Independence High School CANADA $8
1776 Educational Park Drive
San Jose, CA 95133

Published every two months, this is the newsletter of California's computer-using educators, an enthusiastic group of several hundred elementary and high school teachers who have just started an educational software exchange. Members can buy several diskettes of modest educational programs for the Apple, PET, and TRS-80 for $10 a diskette. The newsletter outlines CUE's many activities, and prints short but revealing software reviews from a teacher's viewpoint. It is a model for computer-using educators' groups and is well worth the low membership fee.

EDUCATIONAL COMPUTER MAGAZINE U.S.A. $12
Box 535 CANADA $20
Cupertino, CA 95015

Educational Computer is a new magazine for teachers in schools, colleges, and universities. It is published every two months and features articles, reviews, editorials, and letters on educational topics.

EDUCATIONAL TECHNOLOGY U.S.A. $49
140 Sylvan Avenue CANADA $59
Englewood Cliffs, NJ 07632

A monthly periodical for audio-visual specialists and educators interested in the technological aspects of education. It consists mostly of indifferently edited research papers on arcane subjects written in academic language by professors who must, even at the cost of clear expression, publish or perish. A noticeable exception is Gerald T. Gleason's survey of the use of microcomputers in education (March 1981, pp. 7–18), which succinctly summarizes recent developments in the field. The editors of *Educational Technology* have recently discovered microcomputers and are devoting more and more coverage to their educational applications. This no doubt will be a boon to graduate students and the many education faculty members who judge scholarship by the number of footnotes per manuscript page.

ELECTRONIC LEARNING USA $19
Scholastic Inc.
Box 2001
Englewood Cliffs, NJ 07632

Electronic Learning is a colorful, easy-to-read magazine published eight times a year for teachers. Articles in recent issues include a nine-part primer on computers, a detailed outline of a course in computer literacy, a guide to purchasing microcomputers, and a tutorial on evaluating educational software. It regularly features articles on the educational potential of new products like the videodisc, a comprehensive directory of software houses, and lists of microcomputer courses offered in the United States. Reviews of educational software are short and often superficial, but teachers will find it a good source of ideas for classroom projects. It does not favor any brand of microcomputer. Every elementary and high school staff room should have a subscription.

ILLINOIS SERIES ON THE EDUCATIONAL APPLICATIONS OF COMPUTERS U.S.A. 50¢ per paper
College of Education
University of Illinois
Urbana, IL 61801

A series of academic papers "prepared as resources for the pre-service training of teachers under the general theme of teaching with or about computers." I'm not well enough informed in the teacher-training field to evaluate this comprehensive series. Some of the material may be dated, but no doubt it will be very useful to education faculty interested in adding computer-assisted instructional technology to their curriculums.

INFOWORLD U.S.A. $25
Circulation Department CANADA $52
375 Cochituate Road
Framingham, MA 01701

A weekly newspaper aimed at the personal and business microcomputer user that will keep you informed on the latest developments in the industry. The coverage is comprehensive and most articles are free of jargon. Particularly valuable are its editorials and its many candid software reviews, many of which are for the Apple. Each issue has an extensive collection of classified ads as well as a delightful satire written by "Minnie Floppy." Also revealing are its letters to the editor from irate people who have found that some of the much heralded microcomputer products leave a great deal to be desired. This is the periodical that I look forward to receiving the most.

ITMA NEWSLETTER Free, but send
College of St. Mark and St. John $30 for postage
Derriford Road
Plymouth, PL6 8BH
Great Britain

The quarterly newsletter of the British Investigations on Teaching with Microcomputers as an Aid, a rapidly growing group of British teachers dedicated to developing, evaluating, and promoting CAI materials. It features valuable material on CAI techniques, articles written by teachers, and educational program listings in 380Z BASIC. The newsletter furnishes proof of Britain's energetic promotion of the educational applications of microcomputers.

JOURNAL OF COMPUTER-BASED INSTRUCTION *See* ADCIS
Computer Center
Western Washington University
Bellingham, WA 98225

The *JCBI* is the academic quarterly of Western Washington's Association for the Development of

Computer-Based Instructional Systems. The people who put it out assert that it publishes "original investigations and theoretical papers dealing with direct applications of computing to the problems of learning and instruction, design of curriculum, authoring languages and systems, and comparative curriculum structures for computer-based instruction." Contributors are urged to submit "empirical studies that use experimental procedures which will maximize the potential generalizability of outcomes." Put *that* into your computer.

JOURNAL OF COMPUTERS IN SCIENCE TEACHING U.S.A. $7
Box 4825
Austin, TX 78765

Published quarterly by the Association for Computers in Science Teaching, this journal features research reports, tutorials, and software reviews pertinent to elementary and high school science teaching.

JOURNAL OF COURSEWARE REVIEW U.S.A. $6.95
Box 28426
San Jose, CA 95159

A collection of professional reviews of educational software for the Apple. It is published by the Foundation for the Advancement of Computer-aided Instruction (formerly the Apple Education Foundation), a non-profit organization that furnishes hundreds of thousands of dollars worth of microcomputer equipment each year to people who propose innovative CAI projects. The first issue is $6.95, a modest price to pay to save you from purchasing poor software.

MICRO U.S.A. $24
Box 6502 CANADA $27
Chelmsford, MA 01824

A monthly for experienced APPLE, PET, OSI, TRS-80 Color, and KYM/SYN/AIM programmers. Each issue includes one or more useful utility programs for the Apple, but most of the content is devoted to the other microcomputers. However, *Micro* does have two features that make it worth the subscription price: a monthly annotated bibliography of articles in many microcomputer journals and a comprehensive directory of new software.

MICROCOMPUTING U.S.A. $25
Box 977 CANADA $27
Farmingdale, NY 11737

Microcomputing is a comprehensive, carefully edited monthly noted for excellent articles on technical topics and for regular features on the microcomputer industry, education, business, new products, and book reviews. Serious microcomputer users may prefer it to *Creative Computing*. Editor Wayne Green's perceptive monthly analysis of the industry is worth the price of *Microcomputing*. Highly recommended.

NIBBLE U.S.A. $17.50
Box 325 CANADA $18.00
Lincoln, MA 01733

A magazine for advanced Apple users. Each of the eight yearly issues features at least two major program listings for home, small-business, or entertainment use that can be typed into the Apple. It also contains a selection of program tips, hardware construction projects, and product reviews. The major listings may also be obtained on diskettes for $15 or less. Those who enjoy programming swear by *Nibble*, and no Apple owner should be without a subscription. Highly recommended.

PEELINGS II U.S.A. $21.00
2260 Oleander Street CANADA $28.50
Las Cruces, NM 88004

A privately published collection of comprehensive reviews of Apple II software. Reviewers Edward Burlbaw, Howard de St. Germain, John Metallaro, and John Mitchener don't mince words: if the program is a lemon, they will tell you. What's more, they provide information on the capabilities of software that other reviewers miss. Each program is given a letter grade for any comparison, and advertising is accepted only from companies who are making quality software. *Peelings II* is undoubtedly the most useful single reference for the Apple owner. The review of word processing software in the July issue is itself worth a two-year subscription. It is published nine times a year.

PERSONAL COMPUTERS U.S.A. $1.75 an issue
1 Fawcett Place
Greenwich, CT 06830

Mechanix Illustrated's quarterly magazine on microcomputers. It is designed for newsstand sale (subscription information wasn't given in the first two

issues) and the articles and product reviews are brief and easy to read, but rather superficial.

PERSONAL COMPUTING U.S.A. $18
Circulation Department CANADA $26
1050 Commonwealth Avenue
Boston, MA 02215

A well-designed, easy-to-read montly with articles on varied topics and regular columns on business, education, computer chess, computer bridge, and the future of computing. Since it used to have a slight bias toward the TRS-80, it was originally not high on my recommended list for Apple owners; but recent issues have improved so much under editor David Gabel that *Personal Computing* may soon rival the leaders.

PIPELINE U.S.A. $25 membership
Conduit
Box 388
Iowa City, IA 52244

A semi-annual report of the University of Iowa's CONDUIT organization, a U.S.A. government-supported project designed to develop, evaluate, and market computer-assisted instructional materials for higher education. Members of CONDUIT receive *Pipeline*, a CAI authoring guide, and brochures on new post-secondary CAI materials, some of which are available for the Apple, PET, and TRS-80. *Pipeline* contains several short articles on CAI research, as well as a catalogue of $50 disk-based programs in biology, chemistry, economics, education, geography, psychology, sociology, humanities, management, mathematics, physics, political science, and statistics. Recommended.

POPULAR COMPUTING U.S.A. $15
(formerly ONCOMPUTING) CANADA $18
70 Main Street
Peterborough, NH 03458

McGraw-Hill's new monthly is aimed at magazine-stand browsers and new microcomputer owners. It contains easy-to-read reviews and articles on microcomputers, peripheral devices, software, and new products—all of which are lavishly illustrated with color photographs and diagrams. It is probably the best magazine for beginners.

SCHOOL MICROWARE DIRECTORY U.S.A. $20
Dresden Associates CANADA $20
Box 246
Dresden, MA 04342

A typewritten quarterly catalogue of educational software available for the major microcomputers. Entries are listed by subject and grade level ranging from Kindergarten to Grade 12. Unfortunately, the programs are not reviewed, and anyone purchasing educational software sight unseen can expect to be disappointed.

SCHOOL MICROWARE REVIEW U.S.A. $30
Dresden Associates CANADA $30
Box 246
Dresden, MA 04342

For $30 more than the cost of their preceding directory, Dresden Associates will send you two issues of educational software reviews solicited from subscribers. One would expect such reviews to appear in Dresden's *School Microware Directory*; thus, the publication of a separate newsletter appears to be an attempt to separate educators from their limited computer funds.

SCIENTIFIC AMERICAN U.S.A. $21
Box 5959
New York, NY 10017

This leading scientific monthly occasionally publishes highly technical but well-illustrated articles on microelectronics that are worth searching out. Silicon-chip technology was thoroughly explained in the issues of May 1975 and September 1977; the super-conducting Josephson Junction was featured in the May 1980 issue; disk-storage technology was described in the August 1980 issue; and the new "supercomputers" were discussed in the January 1982 issue.

SOFTALK U.S.A. $24
11021 Magnolia Boulevard CANADA $24
North Hollywood, CA 91601

An excellent monthly that features chatty articles on the people in the microcomputer industry, a lively readers' forum, a programming contest page that will encourage submissions from Apple users of all ages, a revealing disk-jockey-style list of best-selling programs for the Apple. Highly recommended.

SOFTSIDE U.S.A. $24
Box 68 CANADA $32
Milford, NH 03055

A monthly devoted largely to printed-games software for the Apple, PET, TRS-80, and Atari microcomputers. It also has tips for advanced programmers.

TALMIS NEWSLETTER
115 North Oak Park Avenue
Oak Park, IL 60301

A Neilson-rating-style newsletter of educational software reviews scheduled to begin publication in August 1982.

TECHNOLOGY ILLUSTRATED U.S.A. $9.95
Box 2804 CANADA $15.00
Boulder, CO 80321

A lavishly illustrated new magazine designed to explain technological developments to the layman. Like *Science Digest*, the magazine it imitates, *Technology Illustrated* covers a wide range of topics. The first issue (October/November 1981) starts with an article on the history of computers and concludes with a note on the zany wartime research of B. F. Skinner, who is described as "the best-known scientist of our time." I liked it. The photographs were ideal for my introductory lecture on the history of computing, and I still consider B. F. Skinner's *Beyond Freedom and Dignity* to be one of the most important books ever written, even if he did get his start trying to train pigeons to operate the controls of a guided missile. Teachers will probably find *Technology Illustrated* a good source of classroom ideas.

T.H.E. JOURNAL U.S.A. $15
Box 992 CANADA $23
Acton, MD 20172

A journal published every two months subtitled *Technology Horizons in Education* for school administrators by Information Synergy Inc. of Acton, Maryland. It contains short reviews of new equipment and erudite articles on the theoretical and practical applications of technological advances in education at all levels. Complimentary subscriptions are available to senior administrators. I recommend it for educational planners and graduate students in education.

WASHINGTON APPLE PI
Box 34511
Bethesda, MD 20817

The monthly newsletter of the large Apple users group in the Washington, D.C. area. A collection of their best articles 1980–1981 is available for $7.50.

Index

AN APPLE FOR THE TEACHER

Fundamentals of Instructional Computing

George Culp Herbert Nickles

37 B 0300
ISBN 0-534-01378-3